ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information for your TRS-80

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME 8

*Trademark of Tandy Corp.

ENCYCLOPEDIA for the TRS-80*

ENCYCLOPEDIA for the TRS-80*

VOLUME 8

wayne GREEN inc.
PETERBOROUGH NH 03458

*Trademarks of Radio Shack Division of Tandy Corp.

FIRST EDITION
FIRST PRINTING MAY 1982

Edited by Kate Comiskey and Katherine Putnam
Proofread by Ann Winsor
Production: Margaret Baker, Gary Ciocci,
Linda Drew, Thomas Villeneuve, Robert M. Villeneuve,
John R. Schweigert, Sandra Dukette, Karen Stewart
Technical Assistance by Jake Commander and Jim Heid
Illustrations by Howard Happ

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia —a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

Encyclopedia Loader™

The editors of Wayne Green Books want to help you maximize your microcomputing time, so they created the **Encyclopedia Loader™**.

The **Encyclopedia Loader** is a special series of cassettes that offer the longer programs in the **Encyclopedia for the TRS-80*** in ready-to-load form. Each of the ten volumes of the Encyclopedia provides the essential documentation for the programs on the Loader.

With the **Encyclopedia Loader**, you'll save hours of keyboard time and eliminate the aggravating search for typos. The **Encyclopedia Loader** for Volume 8 will contain the programs for the following articles:

Business Predictions from the TRS-80
Stock Valuation
Practical Applications of Classroom Programs
Super Curve Fit
Queen Rama's Cave
TRS-80 Jukebox
Instant Graphics for Everyone
Screen Editor for Graphics Creations
Generate
More Patches to EDTASM

Encyclopedia Loader™ for Volume 1	**EL8001**	**$14.95**
Encyclopedia Loader™ for Volume 2	**EL8002**	**$14.95**
Encyclopedia Loader™ for Volume 3	**EL8003**	**$14.95**
Encyclopedia Loader™ for Volume 4	**EL8004**	**$14.95**
Encyclopedia Loader™ for Volume 5	**EL8005**	**$14.95**
Encyclopedia Loader™ for Volume 6	**EL8006**	**$14.95**
Encyclopedia Loader™ for Volume 7	**EL8007**	**$14.95**
Encyclopedia Loader™ for Volume 8	**EL8008**	**$14.95**

(Please add $1.50 per package for postage & handling)

Mail your order to "Encyclopedia Loader Sales," Wayne Green Books, Pine Street, Peterborough, NH 03458 or call **(1-800-258-5473)**.

*TRS-80 is a trademark of Radio Shack Division of Tandy Corp.

BUSINESS

Business Predictions from the TRS-80
Stock Valuation

Business Predictions from the TRS-80

by Jerry W. O'Dell

There is a mathematical technique called regression analysis which allows you to make stock market predictions. On a TRS-80 microcomputer, working out the predictions is very simple.

Assume that we have data on three stocks: Xerox, IBM, and American Motors, as shown in Table 1. I chose Xerox and IBM because they grew and American Motors because of its poor performance. We will make our predictions from information that is easily available from public records—salaries of manufacturing workers in New York City. The question now is whether we can predict future stock performance from past stock performance. We'll take the 1958–1966 values and try to predict the 1968 value. The mathematics of regression analysis are complex; so I have not gone into detail. If you want to look into the subject, see a text such as Elam McElroy's *Applied Business Statistics* (Holden-Day, 1971), Chapter 9.

Date	Manufacturing Wage	Xerox	IBM	American Motors
1958	$2.13	$1.00	$45.00	$11.00
1960	$2.26	$2.00	$90.00	$25.00
1962	$2.38	$10.00	$100.00	$15.00
1964	$2.54	$35.00	$150.00	$15.00
1966	$2.69	$70.00	$170.00	$10.00
1968	$2.98	$95.00	$325.00	$12.00

Source: U.S. Bureau of Labor Statistics, *Employment and Earnings Statistics for States and Areas*, 1939/72, and *Moody's Handbook*, 1970. Splits were ignored.

Table 1. *Basic data*

You only have to worry about two things in the program, although it is full of comments for the person who wants to delve into it. In line 10, you have to put in the proper number of observations. In the listing, it is five. At the end of the program, you have to enter the data as shown. When you run the program, it develops a prediction formula from the first five values, predicting Xerox stock prices from manufacturing wages. The output is shown in Figure 1. The output is easy to interpret when you get used to it. "Correlation" is a statistical term which refers to the similarity of the values

or how well you can predict from one thing to another. Correlations range from 0 to 1, with higher values indicating greater similarity; a backward relationship is indicated by negative values.

```
CORRELATION          .939476
PREDICTED VALUE OF Y IS  X TIMES 124.368 +  -274.883

   OBS        X          Y       PREDICTED        DIFF
 1.0000    2.1300     1.0000       -9.9793      -10.9793
 2.0000    2.2600     2.0000        6.1885        4.1885
 3.0000    2.3800    10.0000       21.1126       11.1126
 4.0000    2.5400    35.0000       41.0115        6.0115
 5.0000    2.6900    70.0000       59.6666      -10.3334
AVERAGE DIFFERENCE IS      8.99439
ENTER NEW VALUE FOR PREDICTION          2.98
NEW PREDICTED VALUE IS                 95.7333
```

Figure 1. *Xerox data*

The value here is .939476, which is quite high. Manufacturing wages and Xerox stock prices are closely related. The next line gives the formula for predicting Xerox stock prices from manufacturing wages. Take the manufacturing wage number, multiply by 124.368, and subtract 274.883.

The next section of the output actually makes predictions from the data of the first five years. Look at the output. For the first observation, manufacturing wages (X) are $2.13 per hour. To predict the value of Xerox stock in 1958, take 2.13, multiply by 124.368, and subtract 274.833. You get an answer of −9.9793. You didn't get 1.0000, the original Xerox stock value, because there is some error in the prediction. Remember that the correlation is not 1.0000, but .939476.

The last column gives the difference between the actual Xerox values and the predicted Xerox values. You can see how close the prediction is. By themselves, the differences are hard to interpret; so an average is provided in the line below the prediction table. Statisticians use an odd sort of average for this. You can't add the values in the DIFF column and divide by 5; so more complicated means are used.

The average difference is 8.99439, or $8.99 per share. This means that we can predict Xerox scores with an accuracy of about $9. Estimating stocks this accurately could make you a lot of money.

Enter the sixth manufacturing wage value where it says ENTER NEW VALUE FOR PREDICTION. Type in 2.98, the 1968 value. The computer instantly says 95.7333, or $95.73. The predicted value of Xerox stock in 1968 is $95.73; the actual value from Moody's was $95.00. You won't have this

kind of luck all the time but you can prove mathematically that the sort of prediction this program provides is the best that you can do with a simple system of prediction.

If you put the IBM data in the DATA statements with the BASIC editor and run the program, you get the output shown in Figure 2. With IBM, the relationship is even higher. The correlation is .986161, based on the first five values. This is just about as high as you can get. We should expect really good prediction from the IBM data, using the formula shown in the second line. But of course, the TRS-80 works out all the values for us in the third part. Notice that the average difference is approximately 7.38902, or $7.39 per share. In other words, the IBM stock, which sells for at least twice as much as the Xerox stock, can be predicted with even greater accuracy. The computer predicts a 1966 value of $175.24, and the actual value was $170.

```
CORRELATION          .986161
PREDICTED VALUE OF Y IS  X TIMES 221.523  +  -420.654

   OBS        X          Y        PREDICTED        DIFF
 1.0000    2.1300    45.0000        51.1888         6.1888
 2.0000    2.2600    90.0000        79.9868       -10.0132
 3.0000    2.3800   100.0000       106.5690         6.5695
 4.0000    2.5400   150.0000       142.0130        - 7.9869
 5.0000    2.6900   170.0000       175.2420         5.2415
AVERAGE DIFFERENCE IS      7.38902
ENTER NEW VALUE FOR PREDICTION          2.98
NEW PREDICTED VALUE IS                239.483
```

Figure 2. *IBM data*

The real test, of course, is the prediction of the 1968 value. If you enter 2.98 when the computer asks ENTER NEW VALUE FOR PREDICTION, the TRS-80 answers with 239.483, or $239.48. We do not do as well here, for IBM stock sold for $325 that year. Unfortunately for our simple linear program, IBM stock took a huge jump in 1968. But I'll bet that a smart investor would have bought IBM in 1966, knowing there would be a gain of $69.48 ($239.48–$170) in two years.

Xerox and IBM are stocks which did exceptionally well. I've included data in Table 1 for American Motors, a company which is seemingly always in trouble. Put that data into the computer, and you get the results in Figure 3.

Prediction is much worse in this case. American Motors was going down steadily. Remember that backward or falling relationships are shown by a minus correlation. Note that the correlation for American Motors is −.336423. The squared differences are 4.99729 on the average. This seems

better than the previous data, but remember that American Motors stock is much less expensive than the others.

If we make a prediction for 1968 by entering 2.98 when the TRS-80 asks for it, we find that the computer predicts $9.98. In reality, the stock was at $12. A correlation of .336, positive or negative, is not much of a correlation, and the prediction from such a value can only be rough. So you see, with stocks providing high positive correlations, you can safely predict that they are going to continue to rise as the computer shows. Stocks of questionable character show up rapidly, and those with negative correlations are what Moody calls speculative.

```
CORRELATION          -.336423
PREDICTED VALUE OF Y IS  X TIMES  -9.00334  + 36.808

   OBS        X          Y       PREDICTED       DIFF
  1.0000   2.1300    11.0000      17.6309        6.6309
  2.0000   2.2600    25.0000      16.4605       -8.5395
  3.0000   2.3800    15.0000      15.3801        0.3801
  4.0000   2.5400    15.0000      13.9395       -1.0605
  5.0000   2.6900    10.0000      12.5890        2.5890
AVERAGE DIFFERENCE IS     4.99729
ENTER NEW VALUE FOR PREDICTION          2.98
NEW PREDICTED VALUE IS                  9.97807
```

Figure 3. *American Motors data*

It is a good idea to plot the points, say for manufacturing wages and Xerox. A plot gives you a chance to see unusual characteristics of the curve. With correlations of the sort used in this program, the closer the curve is to a straight line, the higher the correlation. Your choice of X, or as statisticians call it, the independent variable, is extremely important. I used manufacturing wages in New York because the data was readily available. There are probably much better predictors. Notice that inflation pushes up both manufacturing wages and stock prices. You have to be careful that you are not simply predicting the effects of inflation.

Program Listing. *Stock prediction*

Encyclopedia Loader

```
  5 CLS
 10 N = 5 :
    ' NUMBER OF PAIRS
 20 FOR I = 1 TO N :
     ' MAIN LOOP
 30  READ X,Y :
     ' READ DATA
 40  XS = XS + X :
     ' SUM VARIABLES
 50  YS = YS + Y
 60  XQ = XQ + X ↑ 2 :
     ' SQUARE, THEN SUM VARIABLES
 70  YQ = YQ + Y ↑ 2
 80  XY = XY + X * Y :
     ' SUM OF CROSS PRODUCTS
 90  NEXT I :
    ' END OF LOOP
100 XM = XS / N :
    ' GET AVERAGES
110 YM = YS / N
120 VX = XQ - XM * XS :
    ' GET SUMS OF SQUARES
130 VY = YQ - YM * YS
140 DX = SQR(VX / N) :
    ' GET STANDARD DEVIATIONS
150 DY = SQR(VY / N)
160 R = (XY - XM * YS) / SQR(VX * VY) :
    ' CALCULATE CORRELATION
170 PRINT "CORRELATION",R :
    PRINT
180 PY = R * (DY / DX) :
    ' GET REGRESSION COEFFICIENT
190 CY = YM - PY * XM :
    ' GET CONSTANT TERM
200 PRINT "PREDICTED VALUE OF Y IS  X TIMES";PY;" + "; CY
210 PRINT
220 PRINT TAB(6);"OBS"; TAB(17);"X"; TAB(27);"Y"; TAB(32);"PREDICTED
    "; TAB(45);"DIFF"
230 RESTORE :
    ' RESET DATA POINTER
240 FOR I = 1 TO N :
     ' LOOP FOR PREDICTIONS
250  READ X,Y :
     ' REREAD DATA
260  YP = X * PY + CY :
     ' MAKE PREDICTION
270  DF = YP - Y :
     ' DIFF BETWEEN PRED AND REAL
280  SQ = DF ↑ 2 :
     ' SQUARE OF THAT
290  SM = SM + SQ :
     ' SUM OF SQUARED DIFFS
300  PRINT USING "#####.####";I,X,Y,YP,DF :
     ' PRINT STUFF
310  NEXT I :
    ' END OF PREDICTION LOOP
320 PRINT
330 AV = SM / N :
    ' AVERAGE SQUARED VALUE
340 PRINT "AVERAGE DIFFERENCE IS", SQR(AV):
    PRINT
350 INPUT "ENTER NEW VALUE FOR PREDICTION  ";X
360 YP = X * PY + CY :
    ' MAKE PREDICTION
370 PRINT "NEW PREDICTED VALUE IS  ",YP
380 GOTO 350
3000 DATA 2.13,1
```

Program continued

```
3010 DATA 2.26,2
3020 DATA 2.38,10
3030 DATA 2.54,35
3040 DATA 2.69,70
```

Stock Valuation

by Charles B. Steele

The excellent stock portfolio programs by Dex Hart in the articles entitled "Put Your Micro on Wall Street" in *Kilobaud Microcomputing*, July and August 1981, provide a great deal of information in a very concise and useful manner. But, each time you want to update the portfolios with current stock prices, you must list the program and edit the DATA lines to enter the new prices.

In the September 1981 issue of *80 Microcomputing* was an article "Slice and Dice BASIC" by J. S. Schneider. This was a potential solution because his article contains a brief utility which causes DATA lines to be modified by the program itself. Better yet, a few minor changes, and DATA lines are altered by means of INPUT statements.

After experimenting with the utility using DATA lines containing numbers and strings as Mr. Hart's program does, I was convinced the two articles could be combined into one. The result is this program entitled Stock Valuation. (See Program Listing.) It works for a bond portfolio or for a combination of stocks and bonds, but I refer only to stocks in describing it.

Sample Runs

To see what the program does, refer to Figure 1 which shows sample runs containing example stocks and information about them. Portval lists the stocks and compares original cost with current value. Timegain compares the current value with the value the last time you updated prices, whether it was a few days ago or several months ago.

The DATA line modification utility in lines 10000 through 10140 of the Program Listing serves two functions. First, when you update prices, it automatically replaces the oldest prices with the newer ones already in the program. Second, when you enter current prices in response to INPUT statement questions, the program fills the new price DATA line with current prices.

Conversion to TRS-80 BASIC

Mr. Hart's program was written in Microsoft BASIC and requires some changes to convert it to TRS-80 Level II BASIC. Mr. Hart's program, besides being a portfolio managing tool, also illustrates some programming techniques, one of which is the PRINT USING statement. A slight modifica-

Portval.....Portfolio Valuation.....Prices as of 21 Jan 81

	Stock	Date	Shares	Cost	Price	Value	Diff	% Gain
1	Carlisle	29Sep80	160	$9,991	85.0	$13,600	$3,609	36.1
2	Crown Cork	18Mar71	100	2,231	32.0	3,200	969	43.4
3	Humana	7Mar77	900	4,900	74.8	67,320	62,420	1273.9
4	Kysor	18Dec69	200	2,758	10.4	2,080	-678	-24.6
5	Travelers	2Dec68	100	3,511	39.3	3,930	419	11.9
6	IBM	29Jul71	200	12,180	64.4	12,880	700	5.7
7	Amer Tel	7Mar72	175	7,850	52.1	9,118	1,268	16.1
8	Texaco	12Feb80	300	11,700	33.4	10,020	-1,680	-14.4
9	Squibb	8Sep79	150	4,200	27.5	4,125	-75	-1.8
10	TRW	12May75	225	4,185	56.0	12,600	8,415	201.1
11	Sears	7Aug80	350	6,895	17.5	6,125	-770	-11.2
12	Emerson El	12Mar77	175	5,994	37.1	6,492	498	8.3
13	Thomas Ind	10Jun73	250	2,211	11.8	2,950	739	33.4
14	Amer El Pw	14Apr74	200	4,600	16.9	3,380	-1,220	-26.5
Totals				$83,206		$157,820	$74,614	89.7

Timegain Stock Value Change Over Time—New prices as of 21 Jan 81
Old prices as of 30 Dec 80

	Stock	Shares	Old Price	Old Value	New Price	New Value	% Port	Diff	% Gain
1	Carlisle	160	84.0	$13,440	85.0	$13,600	8.6	$160	1.2
2	Crown Cork	100	28.4	2,840	32.0	3,200	2.0	360	12.7
3	Humana	900	71.4	64,260	74.8	67,320	42.7	3,060	4.8
4	Kysor	200	10.6	2,120	10.4	2,080	1.3	-40	-1.9
5	Travelers	100	38.9	3,890	39.3	3,930	2.5	40	1.0
6	IBM	200	67.8	13,560	64.4	12,880	8.2	-680	-5.0
7	Amer Tel	175	47.9	8,383	52.1	9,118	5.8	735	8.8
8	Texaco	300	27.5	8,250	33.4	10,020	6.3	1,770	21.5
9	Squibb	150	25.0	3,750	27.5	4,125	2.6	375	10.0
10	TRW	225	60.1	13,523	56.0	12,600	8.0	-923	-6.8
11	Sears	350	15.1	5,285	17.5	6,125	3.9	840	15.9
12	Emerson El	175	36.5	6,388	37.1	6,492	4.1	105	1.6
13	Thomas Ind	250	12.0	3,000	11.8	2,950	1.9	-50	-1.7
14	Amer El Pw	200	16.0	3,200	16.9	3,380	2.1	180	5.6
Totals				$151,888		$157,820	100.0	$5,932	3.9

Figure 1. *Sample runs*

tion was needed here because TRS-80 BASIC uses percent signs to space for strings, whereas his program uses backward slashes. His program uses

COMMON, CHAIN, and RENUMBER, commands which TRS-80 systems without disk do not have. Rearrangement to a menu type of program eliminates the need for these commands.

On INPUT statements under Microsoft BASIC, if you do not want the ? to appear, you can use a comma instead of a semicolon at the end of the INPUT statement. Try this in TRS-80 BASIC, and a syntax error is your response. Accommodation had to be made in Timegain for the 64-character limit of the TRS-80 screen display, but the hard-copy version is spread out like Mr. Hart's program. After a few other minor changes, my TRS-80 was able to understand Portval and Timegain.

Program Description

In addition to the TRS-80 BASIC modifications, I added such features as an introduction, a menu of choices, and a means of preventing the screen from scrolling if the number of stocks is greater than the screen can display at one time while maintaining column headings. I included brief reminder programs so I could quickly determine what stocks and price dates the program currently contains without listing. Mr. Schneider's DATA line modification program is brought into play under the menu choice UPDATE STOCK PRICES.

Copy and load the program, and you are ready to use it. The following example is a program containing 14 stocks and information about them. Run it a few times and try out the options, then substitute your portfolio information. To do this, follow these steps:

- Change the 14 in line 100 to the number of stocks you are using.
- Delete lines 160 through 290 and lines 390 and 400.
- Enter data about each of your stocks, beginning in line 160. Do this in alphabetical order if you wish. You need to enter, in this order, the name of the stock (the program later limits this to 10 characters), date of purchase, number of shares, and original cost. Use a separate DATA line for each stock. Note that in order to keep the dates flush right on the screen or in hard copy, you need to enclose in quotation marks any dates that have single digits. Put a space between the first quotation mark and the date. See line 180 as an example. In this and other places at which you enter dates, keep the abbreviation of the month name to three characters. When entering bonds rather than stocks, instead of entering the number of shares, enter the face value in hundreds of dollars. For example, enter 50 for a $5000 bond.
- In line 390, enter the current price of each stock in the same sequence as the stocks are listed, beginning with line 160. Round off prices to one place. For example, enter 25 3/4 as 25.8. The final entry is the date of the prices. Use a space between day, month, and year because space is not at a premium as it is for the dates used in line 160 and those following it which contain the initial stock information. In order for the data manipulation

subroutine to have enough room to do its job, leave a series of spaces after the date. 15 or 20 spaces should be enough.

- In line 400, enter prices from a previous period and enter the date plus the spaces. This is the last time you will need to make entries in lines 390 and 400. From this point, they are updated by the program.

Future program modifications may be necessary when you add or remove stocks from your portfolio. If there is a stock dividend or split, change the number of shares. If you buy more of the same stock, put it on a separate DATA line as if it were an entirely new addition. If you have more than one portfolio to monitor, imbed it in another copy of the program.

Running the Program

Be sure to save a copy of the modified program. You don't want to do the data entries again. After you type RUN, the title appears, followed by a brief introductory description of the program. The menu appears as follows:

- Item 1 is a reminder. Enter 1, and immediately you see how many stocks are now in the program and a list of them by name.
- Item 2 is another reminder. It gives the dates of the newest prices and the older ones to which they were compared the last time you updated the program.
- Item 3 runs Mr. Hart's Portval program which compares the original cost of each stock with the latest price in the program. There is an option for hard copy if you want a permanent record.
- Item 4 runs Timegain, Mr. Hart's means of keeping up with what has happened to the investments between two periods of time. You also have the option of hard copy.
- Item 5 is where Mr. Schneider's DATA line manipulation subroutine comes in. If you run items 3 and 4 first, you see stock comparisons based on the last prices entered in the program. With item 5, you can update prices to those of today or of any prior date you choose. The next time you run Portval or Timegain, the comparisons are with the most current prices. Selection of item 5 first displays a warning message that the oldest price information now in the program will disappear. If you wish to proceed, each stock name is displayed with a request for the current price. At this point, if you get the message TOO MUCH DATA, you probably did not put enough spaces at the end of DATA line 400. If the price is the same as the prior one, press ENTER. When all prices have been entered, enter the date of the prices. You have a chance to back off and start over if you see an incorrect entry. If everything is correct, you get the message PRICES HAVE BEEN UPDATED. If you get a TOO MUCH DATA message, you probably did not put enough spaces at the end of DATA line 390. When the prices are updated, you can double-check by asking to see the revised DATA lines displayed, or you can go directly back to the menu and run Portval and

Timegain with the current price and date information you have just entered.

- Item 6 saves the updated program. You must add lines to complete this portion, depending on whether you use cassette, stringy, or disk.
- Item 7 exits the program.

It is easy to check the status of a portfolio at any time. Updating takes only a few minutes with the financial page of the newspaper in front of you. Then you will know exactly whether to smile or frown until the next update.

Program Listing

Some comments about the Program Listing are in order. I tried to retain the variables and format of the programs as they were originally published. Note that one of the principal variables in Mr. Hart's programs is the letter I. Avoid using this letter in programs because of its similarity to the number 1 when listed by some printers. Watch for this when you copy the program. Table 1 lists the program variables.

Some lines could be combined and condensed as Mr. Hart pointed out in his article. He used two methods of presenting PRINT USING as illustrations, and I retained both. In Timegain, the four lines of PRINT USING accomplish what Portval takes 11 lines to accomplish. Both ways are excellent examples of PRINT USING.

In the screen display of Timegain, the columns are close together so that they can display a great deal of information in the confines of the 64-character lines. For the hard-copy version, this display has been opened out more and additional features of PRINT USING have been added for commas at the 1000s and dollar signs for the first and total lines. This is the version shown in Figure 1; so don't be surprised when your screen display looks a little different.

Mr. Schneider's DATA line manipulation subroutine is in lines 10000–10140. This is slightly different from his original version because the DATA lines to be changed in the Stock Valuation program contain both numbers and strings. I did this by redefining P1$ in line 10090 to concatenate the string variable, D1$, which is the date of the prices. This change eliminated the need, as mentioned in Mr. Schneider's article, to clean up the trailing comma in the revised DATA line. As a result, in line 10050, the – 1 is deleted from the original.

For complete flexibility on changing DATA lines, I used the variable LIN in lines 10010 and 10020. In Mr. Schneider's example program, he used a specific line number. If you make this a variable, you can call the subroutine to revise line 400, putting newer information from line 390 in the old DATA line, then revise line 390, putting current information in the new DATA line.

You can adapt this subroutine for use in any program in which the DATA lines require frequent change. Its significance as a valuable addition to the

Major Importance

A$	Stock name
C	Cost
D	Difference (value vs cost)
D$	Date of purchase
D1$	Date of new prices
D2$	Date of old prices
G	Gain percentage
I	Array subscript
NUM	Number of stocks
P	New price
P1	Old price
R	Total portfolio percentage
S	Number of shares
T1	Total cost—Portval and Timegain
T2	Total value—Portval and Timegain
T3	Total difference—Portval; total portfolio percentage—Timegain
T4	Total percentage gain—Portval; total difference—Timegain
T5	Total percentage gain—Timegain
V	Current value

Minor Importance and Housekeeping

C	Continue (INKEY$)
C$	Correction of update
H	Hard copy initiation
I$	INKEY$ equivalent
K	VAL(INKEY$)
LC	Line count—stop scroll routine
LIN	Line number to be changed
LL	Print locator
M	Menu recall (INKEY$)
P	Print hard copy—start
U	Update decision
W	Want which choice
X	Counter—FOR-NEXT loops
Y	Counter—FOR-NEXT loops also Yes/No decisions
Z	Print locator

Internal—Data Manipulation

AV
E
L
LN
N
P$
P1$
Q

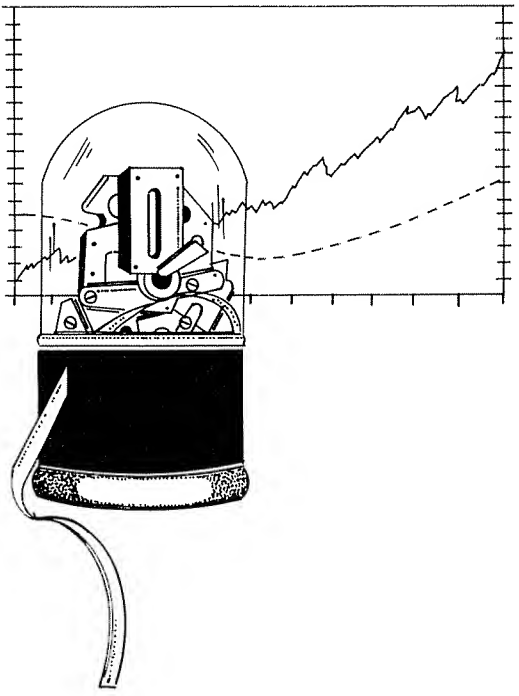
Table 1. *Stock valuation variables*

TRS-80 BASIC toolbox should not be overlooked. I could have used the subroutine to revise the DATA lines containing stock names and costs, but that seems unnecessarily complicated for the occasional addition of a new stock or removal of an old one. I am sure this could be done with INPUT statements, using a modification of the subroutine because of a different sequence of strings and numbers.

Final Notes

Thanks to Mr. Hart and Mr. Schneider, Stock Valuation gives a versatile and uncomplicated portfolio record. It also illustrates DATA line manipulation and formatting with PRINT USING to get a maximum amount of information in a limited space.

In the October 1981 issue of *Kilobaud Microcomputing* Mr. Hart presents a program called CPIndex in the third part of "Put Your Micro on Wall Street." It is a means of comparing how stocks in a portfolio have performed when inflation, as measured by the consumer price index, is taken into account. CPIndex could easily be added to the Stock Valuation program presented here and would make an interesting addition to the menu. If you choose to do so, use the correct variable names to be consistent with those in Stock Valuation.



Program Listing. *Stock Valuation*

Encyclopedia Loader

```
 4 REM  * * * Introduction
 5 CLS
10 PRINT CHR$(23):
   PRINT @320,"S T O C K   V A L U A T I O N"
15 FOR X = 0 TO 300:
    NEXT X:
   Z = 26:
   FOR Y = 452 TO 980 STEP 132
20  PRINT @Y, STRING$(Z,"$");
25  Z = Z - 4:
    FOR X = 0 TO 300:
     NEXT X
30  NEXT Y:
   CLS
40 PRINT :
   PRINT "THIS IS A PROGRAM TO EVALUATE A STOCK PORTFOLIO COMPARING
    ORIG- INAL COST WITH CURRENT VALUE OR COMPARING PRIOR PERIOD VA
   LUE    WITH CURRENT VALUE."
50 PRINT :
   PRINT "IT IS BASED ON A COMBINATION OF 'PORTVAL--TIMEGAIN' BY DE
   X HART (KILOBAUD MICROCOMPUTING, JULY & AUGUST 1981) WITH A DATA
    LINE  MANIPULATION PROGRAM, 'SLICE AND DICE', BY J.S. SCHNEIDER
          (80 MICROCOMPUTING, SEPT. 1981).
60 PRINT :
   PRINT "RUN THIS SAMPLE PROGRAM TO SEE HOW IT WORKS, THEN ENTER Y
   OUR OWNPORTFOLIO IN PLACE OF THE DATA LINES IN THIS SAMPLE.
70 PRINT :
   PRINT "PRESS 'C' TO CONTINUE"
80 IF INKEY$ < > "C"
    THEN
     80
88 :
89 REM  * * * Data load
90 CLEAR 500:
   DEFINT H,I,K,L,X:
   REM  * * * To start program                 without title & intr
   oduction, run from here
100 NUM = 14
110 IF NUM < = 10 GOTO 130
120 DIM A$(NUM),D$(NUM),S(NUM),C(NUM),P(NUM),P1(NUM),V(NUM),D(NUM),G
   (NUM),R(NUM)
130 FOR I = 1 TO NUM
140  READ A$(I),D$(I),S(I),C(I)
150  NEXT I
160 DATA Carlisle, 29Sep80,160,9991
170 DATA Crown Cork,18Mar71,100,2231
180 DATA Humana," 7Mar77",900,4900
190 DATA Kysor,18Dec69,200,2758
200 DATA Travelers," 2Dec68",100,3511
210 DATA IBM,29Jul71,200,12180
220 DATA Amer Tel," 7Mar72",175,7850
230 DATA Texaco,12Feb80,300,11700
240 DATA Squibb," 8Sep79",150,4200
250 DATA TRW,12May75,225,4185
260 DATA Sears," 7Aug80",350,6895
270 DATA Emerson Elect,12Mar77,175,5994
280 DATA Thomas Industries,10Jun73,250,2211
290 DATA Amer El Pwr,14Apr74,200,4600
300 FOR I = 1 TO NUM
310  READ P(I)
320  NEXT I
330 READ D1$:
   REM  * * * Date of "NEW" prices
340 FOR I = 1 TO NUM
350  READ P1(I)
360  NEXT I
370 READ D2$:
```

```
    REM  * * * Date of "OLD" prices
380 D1$ = LEFT$(D1$,9):
    D2$ = LEFT$(D2$,9)
390 DATA 85,32,74.8,10.4,39.3,64.4,52.1,33.4,27.5,56,17.5,37.1,11.8,
    16.9,21 Jan 81
400 DATA 84,28.4,71.4,10.6,38.9,67.8,47.9,27.5,25,60.1,15.1,36.5,12,
    16,30 Dec 80
408 :
409 REM  * * * Menu
410 CLS :
    PRINT :
    PRINT TAB(21);"PROGRAM   SELECTIONS":
    PRINT TAB(17); STRING$(27,".")
420 PRINT TAB(15)"1  REMINDER--LIST OF STOCKS IN PROGRAM"
425 PRINT TAB(15)"2  REMINDER--DATES OF NEW AND OLD PRICES"
430 PRINT TAB(15)"3  RUN 'PORTVAL'"
440 PRINT TAB(15)"4  RUN 'TIMEGAIN'"
450 PRINT TAB(15)"5  UPDATE STOCK PRICES"
460 PRINT TAB(15)"6  SAVE PROGRAM"
470 PRINT TAB(15)"7  END PROGRAM"
480 PRINT @720,"ENTER YOUR SELECTION"
490 K = VAL( INKEY$)
500 IF K < 1 OR K > 7
     THEN
      490 :
     ELSE
      ON K GOTO 1500,1700,2000,3000,4000,4600,4900
1498 :
1499 REM  * * * Reminder-list of stocks in program
1500 CLS :
    PRINT "THERE ARE"NUM"STOCKS NOW IN THE PROGRAM AS FOLLOWS:"
1510 FOR I = 1 TO NUM:
     A$(I) = LEFT$(A$(I),10):
     PRINT I;A$(I),:
     NEXT I
1520 PRINT :
    PRINT :
    PRINT :
    PRINT "TO ADD OR DELETE STOCKS LIST LINES 160-290 AND INSERT OR
    DELETE DATA LINES (IN ALPHABETICAL ORDER, IF DESIRED). THEN, REV
    ISE    VARIABLE 'NUM' IN LINE 100 TO EQUAL NUMBER OF STOCKS."
1530 PRINT TAB(11)"PRESS 'M' TO RETURN TO MENU."
1540 IF INKEY$ = "M"
     THEN
      410 :
     ELSE
      1540
1698 :
1699 REM  * * * Reminder-dates of new and old prices
1700 CLS :
    PRINT :
    PRINT :
    PRINT TAB(13)"DATES OF PRICES NOW IN PROGRAM:"
1710 PRINT :
    PRINT TAB(13)"OLDEST PRICES--AS OF "D2$;
1720 PRINT :
    PRINT TAB(13)"NEWEST PRICES--AS OF "D1$
1730 PRINT :
    PRINT :
    PRINT TAB(13)"PRESS 'M' TO RETURN TO MENU."
1740 IF INKEY$ = "M"
     THEN
      410 :
     ELSE
      1740
1999 :
2000 REM  * * * PORTVAL -- Courtesy of Dex Hart, KILOBAUD MICRO-
         COMPUTING, July & August 1981
2010 CLS :
    PRINT "'PORTVAL'......PORTFOLIO VALUATION......PRICES AS OF ";D1
```

Program continued

```
     $
2020 T1 = 0:
     T2 = 0:
     T3 = 0:
     T4 = 0:
     H = 0:
     LC = 0
2030 FOR I = 1 TO NUM
2040  V(I) = S(I) * P(I)
2050  D(I) = V(I) - C(I)
2060  G(I) = 100 * D(I) / C(I)
2070  T1 = T1 + C(I)
2080  T2 = T2 + V(I)
2090  T3 = T3 + D(I)
2100  T4 = 100 * T3 / T1
2110  NEXT I
2120 PRINT "   STOCK         DATE   SHARES COST  PRICE  VALUE    DIFF
      %GAIN";
2130 FOR I = 1 TO NUM
2140  PRINT USING "## ";I;
2150  PRINT USING "%        % ";A$(I);
2160  PRINT USING "%     %  ";D$(I);
2170  PRINT USING "####   ";S(I);
2180  PRINT USING "#####   ";C(I);
2190  PRINT USING "##.#    ";P(I);
2200  PRINT USING "#####    ";V(I);
2210  PRINT USING "#####  ";D(I);
2220  PRINT USING "####.#";G(I);
2230  LC = LC + 1:
      REM  * * * Line count for stop scrolling
2240  IF LC = 12
       THEN
        LC = - 1:
        GOSUB 2900
2250  NEXT I
2260 IF LC = 11 GOSUB 2900
2270 PRINT STRING$(64,"-");
2280 PRINT "TOTALS";
2290 PRINT TAB(27) USING " ######        ######   ";T1;T2;
2300 PRINT USING "#####   ###.#";T3;T4;
2310 PRINT
2320 GOSUB 5000
2330 IF H < > 1
      THEN
       410
2398 :
2399 REM  * * * Printer start - PORTVAL
2400 CLS :
     PRINT :
     PRINT "IF PRINTER IS ON AND READY WITH PAPER, PRESS 'P' TO START
     .       (TO RETURN TO MENU PRESS 'M'.)"
2410 I$ = INKEY$:
     IF I$ = "M"
      THEN
       410
2420 IF I$ = "P"
      THEN
       2430 :
      ELSE
       2410
2430 CLS :
     PRINT @335,"'PORTVAL' SHOULD BE PRINTING NOW."
2440 GOSUB 12010
2450 GOTO 410
2898 :
2899 REM  * * * Stop screen scrolling - PORTVAL
2900 PRINT "NOTE- INFO IS GOING TO DISAPPEAR; PRESS ANY KEY TO CONTIN
     UE"
2910 IF INKEY$ = "" GOTO 2910 :
      ELSE
```

```
      CLS
2920 PRINT "   STOCK         DATE    SHARES COST  PRICE  VALUE    DIFF
      %GAIN"
2930 RETURN
2999 :
3000 REM  * * * TIMEGAIN -- Courtesy of Dex Hart, KILOBAUD MICRO-
       COMPUTING, August 1981
3010 CLS :
     PRINT " STOCK VALUE CHANGE OVER TIME--NEW PRICES AS OF ";D1$
3020 PRINT TAB(31)"OLD PRICES AS OF ";D2$
3030 T1 = 0:
     T2 = 0:
     T3 = 0:
     T4 = 0:
     T5 = 0:
     H = 0:
     LC = 0
3040 FOR I = 1 TO NUM
3050  C(I) = S(I) * P1(I)
3060  V(I) = S(I) * P(I)
3070  D(I) = V(I) - C(I)
3080  G(I) = 100 * D(I) / C(I)
3090  T1 = T1 + C(I)
3100  T2 = T2 + V(I)
3110  T4 = T4 + D(I)
3120  T5 = 100 * T4 / T1
3130  NEXT I
3140 FOR I = 1 TO NUM
3150  R(I) = 100 * V(I) / T2
3160  T3 = T3 + R(I)
3170  NEXT I
3180 PRINT "                      OLD    OLD    NEW    NEW"
3190 PRINT "    STOCK       SH  PRICE  VALUE  PRICE  VALUE %PORT  DIFF
      %GAIN";
3200 FOR I = 1 TO NUM
3210  PRINT USING "## %        % #### ###.#  #####  ";I;A$(I);S(I);P1
      (I);C(I);
3220  PRINT USING "###.#  ##### ###.# #####  ###.#";P(I);V(I);R(I);D(
      I);G(I);
3230  LC = LC + 1:
      REM  * * * Line count for stop scrolling
3240  IF LC = 10
       THEN
        LC = - 1:
        GOSUB 3900
3250  NEXT I
3260 IF LC = 11 GOSUB 3900
3270 PRINT STRING$(64,"-");
3280 PRINT "TOTALS";
3290 PRINT TAB(25) USING "######        ###### ";T1;T2;
3300 PRINT USING "###.# ##### ####.#";T3;T4;T5;
3310 PRINT
3320 GOSUB 5000
3330 IF H < > 1
      THEN
       410
3398 :
3399 REM  * * * Printer start - TIMEGAIN
3400 CLS :
     PRINT :
     PRINT "IF PRINTER IS ON AND READY WITH PAPER, PRESS 'P' TO START
     .       (TO RETURN TO MENU PRESS 'M'.)"
3410 I$ = INKEY$:
     IF I$ = "M"
      THEN
       410
3420 IF I$ = "P"
      THEN
       3430 :
      ELSE
```

Program continued

```
      3410
3430 CLS :
     PRINT @335,"'TIMEGAIN' SHOULD BE PRINTING NOW."
3440 GOSUB 13010
3450 GOTO 410
3898 :
3899 REM  * * * Stop screen scrolling - TIMEGAIN
3900 PRINT "NOTE- INFO IS GOING TO DISAPPEAR; PRESS ANY KEY TO CONTIN
     UE"
3910 IF INKEY$ = ""
      THEN
       3910 :
      ELSE
       CLS
3920 PRINT "                        OLD    OLD     NEW     NEW"
3930 PRINT "     STOCK       SH  PRICE  VALUE  PRICE   VALUE %PORT  DIFF
       %GAIN";
3940 RETURN
3999 :
4000 REM  * * * Data Line Changes -- Courtesy of J. S. Schneider,
          80-MICROCOMPUTING, September 1981
4010 CLS :
     PRINT :
     PRINT "'UPDATE STOCK PRICES' CAUSES OLDEST PRICES TO BE DELETED
     AND    REPLACED WITH NEWER PRICES ALREADY IN PROGRAM. THEN IT PR
     OVIDES FOR ENTERING CURRENT PRICES.";
4020 PRINT :
     PRINT :
     INPUT "DO YOU WANT TO GO AHEAD WITH THIS(1) OR
           WANT TO RETURN TO MENU(2)";U
4030 CLS :
     IF U < 1 OR U > 2
      THEN
       4020 :
      ELSE
       ON U GOTO 4040,410
4040 CLS :
     PRINT :
     PRINT :
     PRINT "OLD PRICES ARE BEING REPLACED NOW."
4050 LIN = 400:
     GOSUB 10000:
     REM  * * * 400 is line # of oldest         prices
4060 CLS :
     PRINT "STOCK PRICES ARE TO BE BROUGHT UP-TO-DATE NOW.  ENTER CUR
     RENT   PRICE OR PRESS 'ENTER' IF PRICE IS UNCHANGED."
4065 LL = 258:
     PRINT :
     PRINT "WHAT IS THE PRICE OF:"
4070 FOR I = 1 TO NUM:
      A$(I) = LEFT$(A$(I),10):
      REM  * * * Limits stock          name to 10 characters
4080  PRINT @LL,A$(I);:
      PRINT @LL + 11,"";:
      INPUT P(I):
      REM  * * * This          & next line cause stock names & price
      inquries to be           in 3 columns
4085  LL = LL + 21:
      M = M + 1:
      IF M = 3
       THEN
        LL = LL + 1:
        M = 0
4088  NEXT I
4090 INPUT "WHAT IS THE DATE OF THE NEW PRICES"; D1$
4100 PRINT :
     INPUT "ARE THESE PRICES AND THE DATE CORRECT";C$
4110 IF LEFT$(C$,1) < > "Y"
      THEN
       INPUT "PRESS ENTER TO RE-DO";:
```

```
       GOTO 4060
 4120 CLS :
      PRINT :
      PRINT :
      PRINT :
      PRINT "NEW PRICES ARE BEING ENTERED NOW."
 4130 LIN = 390:
      P1$ = "":
      GOSUB 10000:
      REM  * * * 390 is line # of         newer prices
 4140 CLS :
      PRINT :
      PRINT "PRICES HAVE BEEN UPDATED."
 4150 PRINT
 4160 W = 0:
      INPUT "WANT TO RETURN TO MENU (1) OR
             WANT TO SEE PRICE DATA LINES (2)";W
 4170 CLS :
      IF W < 1 OR W > 2
       THEN
        4160 :
       ELSE
        ON W GOTO 410,4180
 4180 PRINT :
      PRINT "AFTER LISTING DATA LINES, TO RETURN TO MENU ENTER 'RUN 90
      '.":
      PRINT
 4190 PRINT :
      PRINT :
      LIST 390 - 400
 4598 :
 4599 REM  * * * Save program
 4600 CLS :
      PRINT :
      PRINT "AT THIS POINT PROGRAM LINES CAN BE ADDED TO SAVE YOUR UPD
      ATED   VERSION.  ADD THE PROPER STATEMENTS TO CAUSE A 'SAVE' DEP
      ENDING ON THE MEDIUM YOU USE (CASSETTE, STRINGY, OR DISK).
 4610 PRINT :
      PRINT "BE SURE TO INCLUDE WARNINGS SUCH AS 'SET UP RECORDER', 'I
      NSERT  CASSETTE', ETC."
 4620 PRINT :
      PRINT "PRESS 'M' TO RETURN TO MENU."
 4630 IF INKEY$ = "M"
       THEN
        410 :
       ELSE
        4630
 4898 :
 4899 REM  * * * Program end
 4900 CLS :
      PRINT :
      PRINT :
      PRINT "TO RETURN TO MENU ENTER 'RUN 90'."
 4910 PRINT :
      PRINT :
      END
 4998 :
 4999 REM  * * * Printer decision
 5000 INPUT "WANT HARD COPY?- - - YES, ENTER 1 (PRINTER ON?); OR TO RE
      TURN TOMENU PRESS 'ENTER'";H:
      RETURN
 9998 :
 9999 REM  * * * Data changes - utility
10000 N = PEEK(17130) * 256 + PEEK(17129)
10010 LN = PEEK(N + 3) * 256 + PEEK(N + 2):
      IF LN > LIN
       THEN
        :
        STOP
10020 IF LN < > LIN
```

Program continued

```
       THEN
        N = PEEK(N + 1) * 256 + PEEK(N):
        GOTO 10010
10030 E = PEEK(N) + 256 * PEEK(N + 1) - 2:
      AV = E - N - 5
10040 P$ = "":
      FOR I = 1 TO NUM:
       P$ = P$ + STR$(P(I)) + ",":
       NEXT
10050 L = LEN(P$)
10060 FOR X = 1 TO L
10070  IF ASC( MID$(P$,X,1)) = 32
        THEN
         10090
10080  P1$ = P1$ + MID$(P$,X,1)
10090  NEXT X:
      P1$ = P1$ + D1$:
      L = LEN(P1$):
      IF L > AV
       THEN
        PRINT "TOO MUCH DATA":
        STOP
10100 FOR X = N + 5 TO E:
       POKE X,32:
       NEXT
10110 FOR X = 1 TO L
10120  Q = ASC( MID$(P1$,X,1))
10130  POKE X + N + 5,Q
10140  NEXT :
      RETURN
12008 :
12009 REM  * * * Hardcopy PORTVAL
12010 LPRINT "'PORTVAL'......Portfolio Valuation......Prices as of ";D
      1$
12020 LPRINT
12120 LPRINT "   Stock         Date   Shares  Cost  Price   Value     D
      iff   %Gain"
12125 LPRINT
12130 FOR I = 1 TO NUM
12140  LPRINT USING "## ";I;
12150  LPRINT USING "%        % ";A$(I);
12160  LPRINT USING "%     %  ";D$(I);
12165  IF I = 1
        THEN
         12570
12170  LPRINT USING "####  ";S(I);
12180  LPRINT USING "##,###  ";C(I);
12190  LPRINT USING "##.#    ";P(I);
12200  LPRINT USING "##,###   ";V(I);
12210  LPRINT USING "##,###  ";D(I);
12220  LPRINT USING "####.#";G(I)
12250  NEXT I
12270 LPRINT STRING$(67,"-")
12280 LPRINT "Totals";
12290 LPRINT TAB(26) USING "$$###,###      $$###,### ";T1;T2;
12300 LPRINT USING "$$##,###   ###.#";T3;T4
12310 RETURN
12570 LPRINT USING "####";S(I);
12580 LPRINT USING "$$##,###  ";C(I);
12590 LPRINT USING "##.# ";P(I);
12600 LPRINT USING "$$##,### ";V(I);
12610 LPRINT USING "$$##,###  ";D(I);
12620 GOTO 12220
13008 :
13009 REM  * * * Hardcopy TIMEGAIN
13010 LPRINT " 'TIMEGAIN' Stock Value Change Over Time--New prices as
      of ";D1$
13020 LPRINT TAB(42)"Old prices as of ";D2$
13030 LPRINT
13180 LPRINT "
```

```
                                        Old      Old     New       New"
13190 LPRINT "     Stock    Shares  Price    Value  Price     Value    %Por
      t      Diff    %Gain"
13195 LPRINT
13200 FOR I = 1 TO NUM
13205  IF I = 1
        THEN
         13510
13210  LPRINT USING "## %         % #### ###.#    ##,###  ";I;A$(I);S(I)
       ;P1(I);C(I);
13220  LPRINT USING "###.#    ##,###    ###.#    ##,###    ###.#";P(I);V(I
       );R(I);D(I);G(I)
13250  NEXT I
13270 LPRINT STRING$(74,"-")
13280 LPRINT "Totals";
13290 LPRINT TAB(24) USING "$$###,###        $$###,###    ";T1;T2;
13300 LPRINT USING "###.# $$##,###  ####.#";T3;T4;T5
13310 RETURN
13510 LPRINT USING "## %         % #### ###.# $$##,###  ";I;A$(I);S(I);
      P1(I);C(I);
13520 LPRINT USING "###.# $$##,###    ###.# $$##,###    ###.#";P(I);V(I)
      ;R(I);D(I);G(I)
13530 GOTO 13250
```

EDUCATION

Practical Applications of Classroom Programs
Super Curve Fit

Practical Applications of Classroom Programs

by Ann Rosenberg

Teaching a high school course in beginning BASIC on the TRS-80 can be fun, exciting, and at the same time challenging.

At first, everything is totally new to the students. Being able to turn on the computer and see READY is a big accomplishment on the first day of class. There are many things the students must learn during the first few months, such as the various types of PRINT statements:

```
PRINT "HELLO"
PRINT A + B
PRINT" THE SUM OF ";A; " + ";B; " = "; A + B
PRINT@480,"USE OF THE VIDEO DISPLAY SHEETS"
PRINT TAB(20):"USE OF TABS"
PRINT CHR$(23);"OH BOY, BIG LETTERS"
```

They also study INPUT statements such as:

```
INPUT R
INPUT"WHAT IS THE RADIUS OF THE CIRCLE";R
```

Assignment statements, READ-DATA, FOR-NEXT, IF-THEN, IF-THEN-ELSE, and ON N GOTO statements soon follow. After discussing each type of statement, the students are able to write different types of assigned programs to emphasize the proper use of each statement.

After the students have mastered not only the programming skills but also the Editor and commands such as LIST, RUN, and DELETE, they are not satisfied with writing plain formula oriented programs.

This is a crucial time in the course. How do you keep students interested and motivated and expand their programming skills without boring them with assigned programs? A graphics assignment is one solution to the problem. After studying SET and RESET, the students are able to use their imagination and creativity to write a program which draws a picture. Most of the students really enjoy this project. Individual students are able to put their personalities into a program and create something totally different from the other students' programs. After completing this project, the students are ready to create programs of their own choosing.

Our school, Metairie Park Country Day, not only has computers in grades 9 through 12, but also utilizes them in kindergarten through grade 6 for drill and enrichment. As part of the high school computer course, each student is required to write a program which can be used in the lower school. This year, the projects were written for the kindergarten, first, and second

grades. The older students talked to the lower school teachers to get ideas and suggestions. When the students finished writing and debugging their programs, we invited several of the children from the lower school to test the programs. I included the following programs in the article.

- Program Listing 1—Dragon Arithmetic. The dragon asks simple addition, subtraction, and multiplication problems. If the student answers correctly, the man kills the dragon, but if he or she is wrong, the dragon kills the man.
- Program Listing 2—Mean Clown. The mean clown starts out with 10 balloons but he pops a random number of balloons each time the program is run. The student can either count the remaining balloons or use subtraction to find the answer.
- Program Listing 3—Compound Words. This program contains 60 compound words. Each time the program is run, 10 words are selected randomly to form a matching test.
- Program Listing 4—Arithmetic for Kids. In this program, the students are asked what level they want to try. Then, they are given 10 problems to answer at that level. If most of their answers are correct, they are rewarded with a smiling face, but if they missed most of the problems, the face cries.

These assignments are not only fun, rewarding, and educational for the computer course students, but also enjoyable and exciting learning experiences for the younger students.

Program Listing 1. *Dragon Arithmetic*

Encyclopedia Loader

```
  1 REM ***** DRAGON *****
  5 REM ***** WRITTEN BY COPEY PULITZER AND MIGUEL URIA *****
 10 CLS :
    RANDOM
 20 PRINT "YOU ARE A SPACEMAN SENT ON A MISSION TO KILL THE DRAGON.
     IF YOUCAN    ANSWER  THE MATH QUESTION GIVEN TO YOU, YOU WILL S
    UCCEED IN YOUR MISSION .  IF YOU FAIL, YOU DIE!!!!"
 25 PRINT :
    INPUT "HOW MANY PROBLEMS DO YOU WISH TO ANSWER";V
 26 PRINT :
    PRINT "WHAT TYPE OF PROBLEM DO YOU WISH TO ANSWER--------":
    INPUT "ADD,SUB, OR MULT.";A$
 27 FOR Z = 1 TO V:
     CLS :
     PRINT "YOU ARE A SPACE MAN SENT ON A MISSION TO KILL THE DRAGON
     . IF YOUCAN ANSWER THE MATH QUESTION GIVEN TO YOU, YOU WILL SUC
     CEED IN  YOUR MISSION. IF YOU FAIL, YOU DIE!!!!"
 28  GOSUB 7000
 31  IF A$ = "ADD"
      THEN
       GOTO 1000
 32  IF A$ = "SUB"
      THEN
       GOTO 2000
 33  IF A$ = "MULT"
      THEN
       GOTO 3000
1000  RANDOM
1010  X = RND(9)
1020  PRINT @296,X;
1030  C = RND(9)
1040  PRINT @360, C;
1050  FOR I = 78 TO 89
1060   SET(I,18):
       NEXT I
1070  PRINT @355, "+";
1080  P = X + C
1100  PRINT @487,;:
      INPUT H
1110  J = H
1200  IF J = P
       THEN
        GOTO 9000
1210  IF J < > P
       THEN
        GOTO 8000
2000  GOTO 2010
2010  X = RND(20)
2020  C = RND(9)
2025  IF C > X
       THEN
        GOTO 2000
2026  IF X < 10 GOTO 2035
2030  PRINT @296,X;
2031  GOTO 2040
2035  PRINT @297,X;
2040  PRINT @361,C;
2050  FOR I = 78 TO 89
2060   SET(I,18):
       NEXT I
2070  PRINT @355,"-";
2080  P = X - C
2090  PRINT @488,;:
      INPUT H:
2100  J = H
2110  IF J = P
       THEN
```

Program continued

```
        GOTO 9000
2120  IF J < > P
       THEN
        GOTO 8000
3000  GOTO 3010
3010  X = RND(9)
3030  PRINT @296,X;
3035  C = RND(9)
3040  PRINT @360,C;
3050  FOR I = 78 TO 89
3060   SET(I,18):
       NEXT I
3070  PRINT @355,"x";
3080  P = X * C
3090  PRINT @487,;:
      INPUT H:
3100  J = H
3110  IF J = P
       THEN
        GOTO 9000
3111  IF J < > P
       THEN
        GOTO 8000
7000  FOR X = 0 TO 127:
       SET(X,38):
       NEXT X
7010  SET(125,33):
      SET(124,34):
      SET(123,34):
      SET(122,35)
7020  FOR X = 118 TO 121:
       SET(X,36):
       NEXT X
7030  SET(119,37):
      SET(120,37)
7040  FOR X = 114 TO 117:
       SET(X,35):
       NEXT X
7050  FOR X = 114 TO 117:
       SET(X,34):
       NEXT X
7060  FOR X = 111 TO 114:
       SET(X,32):
       NEXT X
7070  FOR X = 111 TO 114:
       SET(X,33):
       NEXT X
7080  FOR X = 107 TO 110:
       SET(X,30):
       NEXT X
7090  FOR X = 107 TO 110:
       SET(X,31):
       NEXT X
7100  FOR X = 103 TO 106:
       SET(X,32):
       NEXT X
7110  FOR X = 103 TO 106:
       SET(X,33):
       NEXT X
7120  FOR X = 99 TO 102:
       SET(X,34):
       NEXT X
7130  FOR X = 99 TO 102:
       SET(X,35):
       NEXT X
7140  FOR X = 63 TO 98:
       SET(X,37):
       NEXT X
7150  FOR X = 62 TO 98:
       SET(X,36):
```

```
       NEXT X
7160   FOR X = 61 TO 94:
        SET(X,35):
        NEXT X
7170   FOR X = 60 TO 93:
        SET(X,34):
        NEXT X
7180   FOR X = 59 TO 92:
        SET(X,33):
        NEXT X
7190   FOR X = 58 TO 91:
        SET(X,32):
        NEXT X
7200   RESET(66,32):
       RESET(67,32)
7210   FOR X = 57 TO 90:
        SET(X,31):
        NEXT X
7220   RESET(65,31):
       RESET(66,31):
       RESET(67,31):
       RESET(68,31)
7230   SET(53,31)
7240   FOR X = 46 TO 52:
        SET(X,32):
        NEXT X
7250   FOR X = 46 TO 63:
        SET(X,30):
        NEXT X
7260   FOR X = 70 TO 88:
        SET(X,30):
        NEXT X
7270   FOR X = 46 TO 62:
        SET(X,29):
        NEXT X
7280   FOR X = 71 TO 86:
        SET(X,29):
        NEXT X
7290   FOR X = 46 TO 61:
        SET(X,28):
        NEXT X
7300   RESET(56,28):
       RESET(57,28)
7310   FOR X = 72 TO 85:
        SET(X,28):
        NEXT X
7320   FOR X = 55 TO 60:
        SET(X,27):
        NEXT X
7330   FOR X = 56 TO 58:
        SET(X,27):
        NEXT X
7335   FOR X = 56 TO 58:
        SET(X,26):
        NEXT X
7340   SET(57,25)
7350   SET(3,37):
       SET(7,37):
       SET(4,36):
       SET(6,36)
7360   FOR X = 4 TO 6:
        SET(X,35):
        NEXT X
7370   FOR X = 4 TO 9:
        SET(X,34):
        NEXT X
7380   SET(5,33)
7390   FOR X = 4 TO 6:
        SET(X,32):
        NEXT X
```

Program continued

```
7400  RETURN
8000  FOR X = 52 TO 40 STEP - 1:
       SET(X,31):
       NEXT X
8010  FOR X = 45 TO 40 STEP - 1:
       SET(X,30):
       NEXT X
8020  FOR X = 45 TO 40 STEP - 1:
       SET(X,32):
       NEXT X
8030  FOR X = 43 TO 40 STEP - 1:
       SET(X,33):
       NEXT X
8040  SET(40,34):
      SET(41,34):
      SET(42,35):
      SET(43,35):
      SET(44,36):
      SET(45,36):
      SET(38,29):
      SET(39,29):
      SET(36,28):
      SET(37,28)
8050  SET(38,27):
      SET(39,27)
8060  FOR X = 39 TO 14 STEP - 1:
       SET(X,30):
       NEXT X
8070  FOR X = 39 TO 0 STEP - 1:
       SET(X,31):
       NEXT X
8080  FOR X = 39 TO 0 STEP - 1:
       SET(X,32):
       NEXT X
8090  FOR X = 39 TO 0 STEP - 1:
       SET(X,33):
       NEXT X
8100  FOR X = 37 TO 0 STEP - 1:
       SET(X,34):
       NEXT X
8110  FOR X = 31 TO 0 STEP - 1:
       SET(X,35):
       NEXT X
8120  FOR X = 27 TO 0 STEP - 1:
       SET(X,36):
       NEXT X
8125  GOTO 8140
8130  FOR X = 13 TO 0 STEP - 1:
       SET(X,37):
       NEXT X
8140  FOR X = 30 TO 28 STEP - 1:
       SET(X,29):
       NEXT X
8150  SET(26,28):
      SET(27,28):
      SET(28,27):
      SET(29,27):
      SET(34,35):
      SET(35,35):
      SET(36,36):
      SET(37,36):
      SET(28,37):
      SET(29,37):
      SET(18,27):
      SET(19,27):
      SET(20,28):
      SET(14,29):
      SET(15,29):
      SET(16,28):
      SET(17,28)
```

```
8155  FOR X = 52 TO 40 STEP - 1:
       RESET(X,31):
       NEXT X
8158  FOR X = 45 TO 40 STEP - 1:
       RESET(X,30):
       NEXT X
8161  FOR X = 45 TO 40 STEP - 1:
       RESET(X,32):
       NEXT X
8164  FOR X = 43 TO 40 STEP - 1:
       RESET(X,33):
       NEXT X
8167  RESET(40,34):
      RESET(41,34):
      RESET(42,35):
      RESET(43,35):
      RESET(44,36):
      RESET(45,36):
      RESET(38,29):
      RESET(39,29):
      RESET(36,28):
      RESET(37,28):
      RESET(38,27):
      RESET(39,27)
8170  FOR X = 39 TO 14 STEP - 1:
       RESET(X,30):
       NEXT X
8173  FOR X = 39 TO 0 STEP - 1:
       RESET(X,31):
       NEXT X
8176  FOR X = 39 TO 0 STEP - 1:
       RESET(X,32):
       NEXT X
8179  FOR X = 39 TO 0 STEP - 1:
       RESET(X,33):
       NEXT X
8182  FOR X = 37 TO 0 STEP - 1:
       RESET(X,34):
       NEXT X
8185  FOR X = 31 TO 0 STEP - 1:
       RESET(X,35):
       NEXT X
8188  FOR X = 27 TO 0 STEP - 1:
       RESET(X,36):
       NEXT X
8191  FOR X = 30 TO 28 STEP - 1:
       RESET(X,29):
       NEXT X
8194  RESET(26,28):
      RESET(27,28):
      RESET(28,27):
      RESET(29,27):
      RESET(34,35):
      RESET(35,35):
      RESET(36,36):
      RESET(37,36):
      RESET(28,37):
      RESET(29,37)
8197  RESET(18,27):
      RESET(19,27):
      RESET(20,28):
      RESET(21,28):
      RESET(14,29):
      RESET(15,29):
      RESET(16,28):
      RESET(17,28)
8199  FOR I = 1 TO 300:
       NEXT I:
      RESET(52,30)
8300  FOR I = 1 TO 1000:
```

Program continued

```
       NEXT I
8400   PRINT @832,;"yOU ARE KILLED BY THE DRAGON'S FLAME!!!!!!"
8410   FOR I = 1 TO 500 :
       NEXT I
8500   FOR I = 1 TO 1000:
       NEXT I
8600   CLS
8700   GOTO 9600
9000   FOR X = 12 TO 58:
       SET(X,33):
       NEXT X
9010   FOR X = 1 TO 20:
       NEXT X
9020   FOR X = 12 TO 58:
       RESET(X,33):
       NEXT X
9030   FOR X = 12 TO 58:
       SET(X,33):
       NEXT X
9040   FOR X = 1 TO 20:
       NEXT X
9050   FOR X = 12 TO 58:
       RESET(X,33):
       NEXT X
9060   RESET(57,25):
       FOR X = 56 TO 58:
       RESET(X,26):
       NEXT X
9070   FOR X = 55 TO 60:
       RESET(X,27):
       NEXT X
9080   FOR X = 46 TO 61:
       RESET(X,28):
       NEXT X
9090   FOR X = 46 TO 62:
       RESET(X,29):
       NEXT X
9100   FOR X = 46 TO 63:
       RESET(X,30):
       NEXT X
9110   FOR X = 46 TO 64:
       RESET(X,31):
       NEXT X
9120   FOR X = 46 TO 65:
       RESET(X,32):
       NEXT X
9130   FOR X = 66 TO 56 STEP - 1:
       RESET(X,33):
       NEXT X
9140   FOR X = 66 TO 52 STEP - 1:
       RESET(X,34):
       NEXT X
9150   FOR X = 62 TO 26 STEP - 1:
       SET(X,37):
       NEXT X
9155   FOR X = 61 TO 26 STEP - 1:
       SET(X,36):
       NEXT X
9157   FOR X = 61 TO 26 STEP - 1:
       SET(X,35):
       NEXT X
9160   FOR X = 43 TO 37 STEP - 1:
       SET(X,34):
       NEXT X
9165   FOR X = 40 TO 38 STEP - 1:
       SET(X,33):
       NEXT X
9170   SET(39,32)
9172   RESET(34,35):
       RESET(35,35)
```

```
9175  GOTO 9210
9180  FOR X = 99 TO 117 STEP - 1:
       SET(X,36):
       NEXT X
9190  FOR X = 99 TO 117 STEP - 1:
       SET(X,37):
       NEXT X
9200  SET(118,37):
      SET(119,37):
      SET(121,37):
      SET(123,37):
      SET(125,37):
      SET(126,37):
      SET(120,36):
      SET(122,36):
      SET(124,36)
9209  SET(127,37)
9210  FOR X = 99 TO 127:
       RESET(X,30):
       NEXT X
9211  FOR X = 99 TO 127:
       RESET(X,31):
       NEXT X
9220  FOR X = 99 TO 127:
       RESET(X,32):
       NEXT X
9230  FOR X = 99 TO 127:
       RESET(X,33):
       NEXT X
9240  FOR X = 99 TO 127:
       RESET(X,34):
       NEXT X
9250  FOR X = 99 TO 127:
       RESET(X,35):
       NEXT X
9285  FOR X = 99 TO 102:
       RESET(X,35):
       NEXT X
9287  FOR X = 114 TO 117:
       RESET(X,35):
       NEXT X
9290  FOR X = 99 TO 127:
       RESET(X,36):
       NEXT X
9300  FOR X = 118 TO 124:
       RESET(X,37):
       NEXT X
9400  FOR X = 99 TO 117:
       SET(X,36):
       NEXT X
9459  FOR X = 99 TO 127:
       SET(X,37):
       NEXT X
9500  FOR X = 1 TO 700
9555   PRINT @832,;"YOU HAVE SLAYED THE DRAGON!!!!"
9556   FOR X = 1 TO 1500:
        NEXT X
9600   NEXT Z
9700  CLS
10000 INPUT "WOULD YOU LIKE TO PLAY AGAIN(YES OR NO)";F$
10090 IF F$ < > "YES" AND F$ < > "NO" GOTO 10000
11000 IF F$ = "YES" GOTO 10
```

Program Listing 2. *Mean Clown*

```
 1 REM ***** MEAN CLOWN *****
 5 REM ***** WRITTEN BY CLAIRE LEAMAN AND NOAH SHAPIRA *****
10 REM  GAMES
20 CLS :
   RANDOM
```

Program continued

```
  30 PRINT CHR$(23);"HI! I AM YOUR FRIENDLY COMPUTER.":
     INPUT "WHAT IS YOUR NAME";N$
  35 PRINT
  40 PRINT "HELLO  ";N$:
     PRINT "    WOULD YOU LIKE TO  PLAY SOME GAMES WITH ME (ANSWER YE
     S OR NO AND THEN PRESS ENTER)";:
     INPUT Q$
  50 IF Q$ = "NO"
      THEN
       PRINT CHR$(23);"OKAY "; N$;" SEE YOU LATER":
       END
  60 IF Q$ = "YES"
      THEN
       GOTO 80
  70 PRINT "YOU MUST ANSWER YES OR NO":
     GOTO 40
  80 GOTO 90
  90 CLS
1000 FOR X = 6 TO 15:
      SET (X,6):
      NEXT X:
     FOR Y = 7 TO 9:
      SET(15,Y):
      NEXT Y:
     FOR Y = 7 TO 9:
      SET(14,Y):
      NEXT Y:
     FOR X = 15 TO 6 STEP - 1:
      SET(X,9):
      NEXT X:
     FOR Y = 8 TO 6 STEP - 1:
      SET(6,Y):
      NEXT Y:
     FOR Y = 8 TO 6 STEP - 1:
      SET(7,Y):
      NEXT Y:
     FOR Y = 10 TO 17:
      SET(10,Y):
      NEXT Y
1010 FOR Y = 10 TO 17:
      SET(11,Y):
      NEXT Y:
     FOR X = 18 TO 27:
      SET(X,4):
      NEXT X:
     FOR Y = 5 TO 7:
      SET(26,Y):
      NEXT Y:
     FOR Y = 5 TO 7:
      SET(27,Y):
      NEXT Y:
     FOR X = 25 TO 18 STEP - 1:
      SET(X,7):
      NEXT X:
     FOR Y = 6 TO 5 STEP - 1:
      SET(18,Y):
      NEXT Y:
     FOR Y = 6 TO 5 STEP - 1:
      SET(19,Y):
      NEXT Y
1020 FOR Y = 8 TO 15:
      SET(22,Y):
      SET(23,Y):
      NEXT Y:
     FOR X = 30 TO 39:
      SET(X,6):
      NEXT X:
     FOR Y = 7 TO 9:
      SET(38,Y):
      SET(39,Y):
```

```
      NEXT Y:
     FOR X = 37 TO 30 STEP - 1:
      SET(X,9):
      NEXT X
1030 FOR Y = 8 TO 7 STEP - 1:
      SET(31,Y):
      NEXT Y:
     FOR Y = 8 TO 7 STEP - 1:
      SET(30,Y):
      NEXT Y:
     FOR Y = 10 TO 17:
      SET(34,Y):
      NEXT Y:
     FOR Y = 10 TO 17:
      SET(35,Y):
      NEXT Y
1040 FOR X = 42 TO 51:
      SET (X,8):
      NEXT X:
     FOR Y = 9 TO 11:
      SET(50,Y):
      NEXT Y:
     FOR Y = 9 TO 11:
      SET(51,Y):
      NEXT Y:
     FOR X = 49 TO 42 STEP - 1:
      SET(X,11):
      NEXT X:
     FOR Y = 10 TO 9 STEP - 1:
      SET(42,Y):
      NEXT Y
1050 FOR Y = 10 TO 9 STEP - 1:
      SET(43,Y):
      NEXT Y:
     FOR Y = 12 TO 19:
      SET(46,Y):
      NEXT Y
1060 FOR Y = 12 TO 19:
      SET(47,Y):
      NEXT Y:
     FOR X = 54 TO 63:
      SET(X,6):
      NEXT X:
     FOR Y = 6 TO 9:
      SET(63,Y):
      SET(62,Y):
      NEXT Y:
     FOR X = 63 TO 54 STEP - 1:
      SET(X,9):
      NEXT X:
     FOR Y = 9 TO 6 STEP - 1:
      SET(54,Y):
      SET(55,Y):
      NEXT Y
1070 FOR Y = 10 TO 17:
      SET(58,Y):
      SET(59,Y):
      NEXT Y:
     FOR X = 66 TO 75:
      SET(X,4):
      NEXT X:
     FOR Y = 4 TO 7:
      SET(75,Y):
      SET(74,Y):
      NEXT Y:
     FOR X = 75 TO 66 STEP - 1:
      SET(X,7):
      NEXT X:
     FOR Y = 7 TO 4 STEP - 1:
      SET(66,Y):
```

Program continued

```
      SET(67,Y):
      NEXT Y
1080 FOR Y = 8 TO 15:
      SET(70,Y):
      SET(71,Y):
      NEXT Y:
     FOR X = 78 TO 87:
      SET(X,6):
      NEXT X:
     FOR Y = 6 TO 9:
      SET(87,Y):
      SET(86,Y):
      NEXT Y:
     FOR X = 87 TO 78 STEP - 1:
      SET(X,9):
      NEXT X:
     FOR Y = 9 TO 6 STEP - 1:
      SET(78,Y):
      SET(79,Y):
      NEXT Y
1090 FOR Y = 10 TO 17:
      SET(82,Y):
      SET(83,Y):
      NEXT Y:
     FOR X = 90 TO 99:
      SET(X,8):
      NEXT X:
     FOR Y = 8 TO 11:
      SET(99,Y):
      SET(98,Y):
      NEXT Y:
     FOR X = 99 TO 90 STEP - 1:
      SET(X,11):
      NEXT X:
     FOR Y = 11 TO 8 STEP - 1:
      SET(90,Y):
      SET(91,Y):
      NEXT Y
1100 FOR Y = 12 TO 19:
      SET(94,Y):
      SET(95,Y):
      NEXT Y:
     FOR X = 102 TO 111:
      SET(X,6):
      NEXT X:
     FOR Y = 6 TO 9:
      SET(111,Y):
      SET(110,Y):
      NEXT Y:
     FOR X = 111 TO 102 STEP - 1:
      SET(X,9):
      NEXT X
1110 FOR Y = 9 TO 6 STEP - 1:
      SET(102,Y):
      SET(103,Y):
      NEXT Y:
     FOR Y = 10 TO 17:
      SET(106,Y):
      SET(107,Y):
      NEXT Y:
     FOR X = 114 TO 123:
      SET(X,4):
      NEXT X:
     FOR Y = 4 TO 7:
      SET(123,Y):
      SET (122,Y):
      NEXT Y
1120 FOR X = 123 TO 114 STEP - 1:
      SET(X,7):
      NEXT X:
```

```
     FOR Y = 7 TO 4:
      SET(114,Y):
      SET(115,Y):
      NEXT Y:
     FOR Y = 8 TO 15:
      SET(118,Y):
      SET(119,Y):
      NEXT Y
1130 FOR Y = 7 TO 4 STEP - 1:
      SET(114,Y):
      SET(115,Y):
      NEXT Y:
     SET(12,21):
     SET(13,21):
     SET(16,21):
     SET(17,21):
     SET (11,22):
     SET(12,22):
     SET(17,22):
     SET(18,22):
     SET(10,23):
     SET(11,23):
     SET(18,23):
     SET(19,23)
1140 SET(10,24):
     SET(19,24):
     FOR X = 8 TO 21:
      SET(X,25):
      SET(X,31):
      NEXT X:
     SET(8,26):
     SET(9,26):
     SET(20,26):
     SET(21,26):
     SET(7,27):
     SET(8,27):
     SET(9,27):
     SET(13,27):
     SET(16,27):
     SET(20,27):
     SET(21,27)
1150 SET(22,27):
     FOR X = 6 TO 9:
      SET(X,28):
      NEXT X:
     SET(8,29):
     SET(9,29):
     FOR X = 12 TO 17:
      SET(X,29):
      NEXT X:
     SET(20,29):
     SET(21,29):
     SET(8,30):
     SET(9,30):
     SET(12,30):
     SET(17,30):
     SET(20,30):
     SET(21,30)
1160 FOR X = 20 TO 23 :
      SET(X,28):
      NEXT X:
     SET(14,32):
     SET(15,32):
     FOR X = 6 TO 23:
      SET(X,33):
      SET(X,40):
      NEXT X:
     FOR Y = 32 TO 36:
      SET(2,Y):
```

Program continued

```
       SET(27,Y):
       NEXT Y:
      FOR Y = 34 TO 36:
       SET(3,Y):
       SET(26,Y):
       NEXT Y
 1170 SET(4,36):
      SET(5,36):
      SET(24,36):
      SET(25,36):
      FOR Y = 34 TO 40:
       SET(6,Y):
       SET(7,Y):
       SET(22,Y):
       SET(23,Y):
       NEXT Y:
      FOR Y = 41 TO 44:
       SET(12,Y):
       SET(13,Y):
       SET(16,Y):
       SET(17,Y):
       NEXT Y
 1180 FOR Y = 42 TO 44:
       SET(4,Y):
       SET(25,Y):
       NEXT Y:
      FOR Y = 43 TO 44:
       SET(5,Y):
       SET(24,Y):
       NEXT Y:
      FOR X = 6 TO 11:
       SET(X,44):
       NEXT X:
      FOR X = 18 TO 23:
       SET(X,44):
       NEXT X
 1190 PRINT @773,"MEAN";:
      Z = RND(9) + 1:
      PRINT @590,"IF THE MEAN CLOWN";
 1200 PRINT @654,"POPS";Z;"BALLOONS,";:
      FOR I = 1 TO 10:
       NEXT I:
      M = 127 - (3 + Z * 12):
      FOR X = 127 TO M STEP - 1:
       FOR Y = 3 TO 20:
        RESET(X,Y):
        NEXT Y:
       NEXT X:
      INPUT "HOW MANY BALLOONS ARE LEFT";P
 1210 IF P < > 10 - Z
       THEN
        CLS :
        PRINT "WRONG":
        GOTO 1234
 1220 CLS
 1230 PRINT @450,N$ ", YOU ARE CORRECT!";
 1234 PRINT " WOULD YOU LIKE TO PLAY AGAIN";:
      INPUT L$
 1235 IF L$ = "NO"
       THEN
        PRINT "BYE "; N$; " I HOPE YOU HAD FUN!":
        END
 1240 IF L$ = "YES"
       THEN
        GOTO 90
 1250 PRINT "YOU MUST ANSWER YES OR NO":
      GOTO 1210
 1260 END
```

Program Listing 3. *Compound Words*

```
  1 REM ***** COMPOUND WORDS *****
  5 REM ***** WRITTEN BY ROBYN PERRIN AND KEN COLOMES *****
 10 CLS :
    RANDOM
 12 CLEAR 600
 15 DIM A$(60),B$(60),C$(10),D$(10),E$(10)
 20 PRINT "THERE ARE 2 COLUMNS OF WORDS. EACH WORD FROM THE LEFT COL
    UMN     GOES WITH A WORD IN THE RIGHT COLUMN TO MAKE A COMPOUND
    WORD."
 30 FOR I = 1 TO 60
 40  READ A$(I),B$(I)
 60  NEXT I
 70 N = RND(60)
 80 C$(1) = A$(N):
    D$(1) = B$(N)
 90 FOR X = 2 TO 10
100  N = RND(60)
110  FOR J = 1 TO X - 1
120   IF C$(J) = A$(N)
       THEN
        N = RND(60):
        J = 1:
        GOTO 120
130   NEXT J
140  C$(X) = A$(N):
     D$(X) = B$(N)
150  NEXT X
190 N = RND(10)
200 E$(1) = D$(N)
210 FOR X = 2 TO 10
220  N = RND(10)
230  FOR J = 1 TO X - 1        8                    00/00/00 00:08:07
240   IF E$(J) = D$(N)
       THEN
        N = RND(10):
        J = 1:
        GOTO 240
250   NEXT J
260  E$(X) = D$(N)
270  NEXT X
280 FOR I = 1 TO 10
290  PRINT I;C$(I),I;E$(I)
300  NEXT I
305 FOR J = 1 TO 10
310  PRINT @896, "WHAT GOES WITH #";J;:
     INPUT A:
     PRINT @896,"                                    ":
     IF A > 10
      THEN
       GOTO 310
315  IF E$(A) = D$(J)
      THEN
       PRINT @290,C$(J);D$(J) " IS CORRECT";:
       FOR I = 1 TO 1000:
        NEXT I:
       PRINT @290,"                              ";:
       GOTO 400
316  C = C + 1:
     PRINT @896,"TRY AGAIN":
     FOR Z = 1 TO 400:
      NEXT Z:
     GOTO 310
400  NEXT J
405 PRINT @896,"YOU GOT ";10 - C;" OUT OF 10 RIGHT"
410 PRINT @960,"DO YOU WANT TO DO ANY MORE <YES OR NO>";:
    INPUT Z$
420 IF Z$ = "YES"
```

Program continued

```
      THEN
       GOTO 10
  430 IF Z$ = "NO"
      THEN
       CLS :
       END
10000 DATA AFTER,NOON,WITH,OUT,EVERY,ONE,AIR,PLANE,CLUB,HOUSE,DOOR,WAY
      ,SOME,TIME,EVER,GREEN,NEWS,CAST,SHOE,HORN,WATER,FALL,UNDER,DOG,P
      ASS,PORT,DRAW,BRIDGE,YOUR,SELF,ANY,THING,GENTLE,MAN,BATH,ROOM,BE
      D,ROOM,FIRE,CRACKER,BIRTH,DAY,CAR,PET,BLACK,SMITH,FISHER,MAN
10001 DATA FAIRY,LAND,WORK,BOOK,HOME,WORK,SHOT,GUN,FOOT,BALL,BASKET,BA
      LL,QUARTER,BACK,GOLD,FISH,FROG,MAN,FOUR,TEEN,FOX,HOLE,FOR,TUNE,F
      ORE,HEAD,FOOT,PRINT,SUIT,CASE,BED,SPREAD,NOTE,BOOK,OUT,LINE,OVER
      ,COAT,RAIN,COAT,SAIL,BOAT,BREAK,FAST
10002 DATA NIGHT,GOWN,NICK,NAME,PAD,LOCK,PAPER,BACK,HIGH,WAY,OAT,MEAL,
      SCREW,DRIVER,OUT,SIDE,LAY,OUT,PACK,AGE,SUN,BURN,GRAND,PARENTS,GR
      APE,VINE,FORE,SHADOW,PHOTO,GRAPH
```

Program Listing 4. *Arithmetic for Kids*

```
  1 REM ***** ARITHMETIC FOR KIDS *****
  5 REM ***** WRITTEN BY MARY CYNTHIA DUPY AND
  6 REM ***** MARY VIRGINIA WEINMANN *****
 20 CLS
 30 PRINT "","ADDITION AND SUBTRACTION IN LEVELS"
 31 PRINT :
    PRINT "   THIS PROGRAM HAS THREE DIFFERENT LEVELS OF DIFFICULTY.
     THE    FIRST LEVEL IS FOR BEGINNERS IN MATH. THE SECOND LEVEL I
    S  FOR PEOPLE WHO KNOW A LITTLE BIT OF MATH, BUT AREN'T EXPERTS
    YET.THE THIRD LEVEL IS FOR THOSE LITTLE MATHEMATICAL ";
 32 PRINT "GENIUSES THAT PLAN TO GO ON TO CALCULUS IN LATER LIFE.":
    PRINT "  THERE ARE TEN(10) PROBLEMS IN EACH LEVEL. WHEN YOU GET
    MOST OF THE PROBLEMS RIGHT A SMILEY FACE WILL SHOW ON THE SCREEN
    . WHEN YOU GET MOST OF THEM WRONG A FROWNEY FACE  WILL SHOW."
 40 PRINT "   WHEN YOU ARE READY TO GO ON, JUST PRESS <ENTER>."
 50 INPUT Z
 51 CLS
 60 PRINT :
    PRINT "","DIRECTIONS"
 70 PRINT :
    PRINT :
    PRINT "   WHEN YOU DECIDE WHICH LEVEL YOU WANT TO TRY PRESS THAT
     NUMBER AND THAT PROGRAM WILL BEGIN. IF YOU WANT TO STOP THAT PR
    OGRAM  IN THE MIDDLE THEN PRESS<BREAK>(THEN YOU'LL HAVE TO START
     ALL   OVER AGAIN).";
 71 PRINT " IF YOU WANT TO STOP THE WHOLE THING, PRESS #4."
 80 PRINT :
    PRINT "LEVEL 1> SIMPLE ADDITION(VERTICAL PROBLEMS)":
    PRINT "LEVEL 2> HARDER ADDITION AND SUBTRACTION(VERTICAL AND HOR
    IZONTAL         PROBLEMS)":
    PRINT "LEVEL 3> ADVANCED ADDITION AND SUBTRACTION(HORIZONTAL PRO
    BLEMS)":
    PRINT "      4> STOP"
 90 INPUT "WHICH LEVEL DO YOU WANT";L
100 IF L = 1
     THEN
      GOTO 110
101 IF L = 2
     THEN
      GOTO 210
102 IF L = 3
     THEN
      GOTO 310
103 IF L = 4
     THEN
      CLS :
```

```
      PRINT "","MARY CYNTHIA DUPY":
      PRINT "","          AND",:
      PRINT :
      PRINT "","MARY VIRGINIA WEINMANN":
      FOR Z = 1 TO 1000:
       NEXT Z:
      CLS :
      END
104 PRINT "YOU MUST ANSWER<1,2,3,OR 4>!!!!!":
    GOTO 90
110 CLS :
    PRINT "","THIS IS THE EASIEST PROGRAM"
112 FOR Z = 1 TO 1000:
     NEXT Z:
    CLS
113 C = 0:
    B = 0
115 PRINT
116 FOR T = 1 TO 10
120  N = RND(10) - 1:
     M = RND(10) - 1:
     IF N + M > 10
      THEN
       120
121  A = N + M
130  PRINT @220,N;:
     PRINT @283,"+"M
131  FOR X = 54 TO 63:
      SET(X,15):
      NEXT X
132  PRINT :
     PRINT "                                ";:
     INPUT Q
140  IF Q = (N + M)
      THEN
       PRINT @475,"RIGHT!":
       C = C + 1
141  IF Q < > (N + M)
      THEN
       PRINT @475,"WRONG, THE ANSWER IS "A".":
       B = B + 1
142  FOR Z = 1 TO 1000:
      NEXT Z:
     CLS :
     NEXT T
143 PRINT :
    PRINT :
    PRINT :
    PRINT "YOU GOT ";C;" PROBLEMS RIGHT AND ";B;" PROBLEMS WRONG.":
    FOR Z = 1 TO 750:
     NEXT Z:
    CLS
145 IF C > = 6
     THEN
      GOSUB 10000
147 IF C < 6
     THEN
      GOSUB 20000
148 INPUT "WOULD YOU LIKE TO TRY AGAIN";W$
150 IF W$ = "YES"
     THEN
      GOTO 112
152 IF W$ = "NO"
     THEN
      GOTO 51
155 PRINT "YOU MUST ANSWER <YES> OR <NO>!!!!":
    GOTO 145
210 CLS :
    PRINT "","THIS IS THE HARDER PROGRAM."
215 PRINT :
```

Program continued

```
    C = 0:
    B = 0
216 FOR T = 1 TO 10
218  FOR Z = 1 TO 1000:
      NEXT Z:
     CLS :
     C = 0:
     B = 0
220  N = RND(10) - 1:
     M = RND(10) - 1
221  A = N + M
225  ON RND(4) GOTO 230,232,234,236
230  PRINT @220,N;:
     PRINT @283,"+"M:
     FOR X = 54 TO 63:
      SET(X,15):
      NEXT X:
     PRINT :
     INPUT "                                ";Q:
     GOTO 240
232  PRINT @215,N"+"M"=";:
     INPUT Q:
     GOTO 240
234  IF N - M < 0
      THEN
       N = RND(10) - 1:
       GOTO 234
235  PRINT @220,N;:
     PRINT @283,"-"M:
     FOR X = 54 TO 63:
      SET(X,15):
      NEXT X:
     PRINT :
     INPUT "                                ";R:
     GOTO 252
236  IF N - M < 0
      THEN
       N = RND(10) - 1:
       GOTO 236
237  PRINT @215,N"-"M"=";:
     INPUT R:
     GOTO 252
240  IF Q = N + M
      THEN
       PRINT @475, "RIGHT!":
       C = C + 1:
       GOTO 256
250  IF Q < > N + M
      THEN
       PRINT @467, "WRONG, THE ANSWER IS "A".":
       B = B + 1:
       GOTO 256
252  IF R = N - M
      THEN
       PRINT @475, "RIGHT!";:
       C = C + 1
254  IF R < > N - M
      THEN
       PRINT @467, "WRONG, THE ANSWER IS "N - M".":
       B = B + 1
256  NEXT T
260 PRINT :
    PRINT :
    PRINT :
    PRINT "YOU GOT "C" PROBLEMS RIGHT AND "B" PROBLEMS WRONG.":
    FOR Z = 1 TO 1000:
     NEXT Z:
    CLS
263 IF C > = 6
     THEN
```

```
      GOSUB 1000
265 IF C < 6
     THEN
      GOSUB 2000
270 INPUT "WOULD YOU LIKE TO TRY AGAIN";W$
271 IF W$ = "YES"
     THEN
      GOTO 210
272 IF W$ = "NO"
     THEN
      GOTO 51
273 PRINT "YOU MUST ANSWER <YES> OR <NO>!!!!":
    GOTO 270
310 CLS :
    PRINT "",:
    PRINT "THIS IS THE HARDEST PROGRAM."
320 FOR Z = 1 TO 1000:
     NEXT Z:
    CLS
330 PRINT :
    C = 0:
    B = 0
340 FOR T = 1 TO 10
350  N = RND(20) - 1:
     M = RND(20) - 1
360  ON RND(2) GOTO 370,380
365  IF N + M > 30
      THEN
       N = RND(20) - 1:
       GOTO 365
370  PRINT @215,N"+"M"=";:
     INPUT P:
     GOTO 400
380  IF N - M < 0
      THEN
       N = RND(20) - 1:
       GOTO 380
390  PRINT @215,N"-"M"=";:
     INPUT P:
     GOTO 420
400  IF P = N + M
      THEN
       PRINT @475,"RIGHT!":
       C = C + 1:
       GOTO 440
410  IF P < > N + M
      THEN
       PRINT @467, "WRONG, THE ANSWER IS"N + M".":
       B = B + 1:
       GOTO 440
420  IF P = N - M
      THEN
       PRINT @475,"RIGHT!":
       C = C + 1:
       GOTO 440
430  IF P < > N - M
      THEN
       PRINT @467,"WRONG, THE ANSWER IS"N - M".":
       B = B + 1
440  FOR Z = 1 TO 750:
      NEXT Z:
     CLS
450  NEXT T
460 PRINT "YOU GOT "C" PROBLEMS RIGHT AND "B" PROBLEMS WRONG."
470 IF C > = 6
     THEN
      GOSUB 1000
480 IF C < 6
     THEN
      GOSUB 2000
```

Program continued

```
 490 INPUT "WOULD YOU LIKE TO TRY AGAIN"; W$
 500 IF W$ = "YES"
      THEN
       GOTO 310
 510 IF W$ = "NO"
      THEN
       GOTO 51
 520 PRINT "YOU MUST ANSWER <YES> OR <NO>!!!":
     GOTO 490
 999 END
1000 FOR X = 28 TO 99:
      SET (X,6):
      NEXT X:
     FOR Y = 6 TO 35:
      SET (98,Y):
      NEXT Y
1010 FOR Y = 6 TO 35:
      SET(99,Y):
      NEXT Y:
     FOR X = 99 TO 28 STEP - 1:
      SET (X,35):
      NEXT X
1020 FOR Y = 35 TO 6 STEP - 1:
      SET(29,Y):
      NEXT Y:
     FOR Y = 35 TO 6 STEP - 1:
      SET(28,Y):
      NEXT Y
1030 FOR X = 44 TO 49:
      SET(X,12):
      NEXT X:
     FOR X = 44 TO 49:
      SET(X,13):
      NEXT X:
     FOR X = 44 TO 49:
      SET(X,14):
      NEXT X
1040 FOR X = 78 TO 83:
      SET(X,12):
      NEXT X:
     FOR X = 78 TO 83:
      SET(X,13):
      NEXT X:
     FOR X = 78 TO 83:
      SET(X,14):
      NEXT X
1050 FOR X = 62 TO 65:
      SET(X,18):
      NEXT X:
     FOR X = 62 TO 65:
      SET(X,19):
      NEXT X:
     FOR X = 62 TO 65:
      SET(X,20):
      NEXT X
1052 FOR T = 1 TO 3:
      FOR Y = 24 TO 27:
       RESET(44,Y):
       NEXT Y:
      FOR Y = 24 TO 27:
       RESET(45,Y):
       NEXT Y:
      FOR Y = 24 TO 27:
       RESET(46,Y):
       NEXT Y:
      FOR Y = 24 TO 27:
       RESET(47,Y):
       NEXT Y:
      FOR X = 48 TO 81:
       RESET(X,26):
```

```
       NEXT X:
      FOR X = 48 TO 81:
       RESET(X,27):
       NEXT X
1053  FOR Y = 27 TO 24 STEP - 1:
       RESET(82,Y):
       NEXT Y:
      FOR Y = 27 TO 24 STEP - 1:
       RESET(83,Y):
       NEXT Y:
      FOR Y = 27 TO 24 STEP - 1:
       RESET(84,Y):
       NEXT Y:
      FOR Y = 27 TO 24 STEP - 1:
       RESET(85,Y):
       NEXT Y
1055  FOR Y = 24 TO 27:
       SET(44,Y):
       NEXT Y:
      FOR Y = 24 TO 27:
       SET(45,Y):
       NEXT Y:
      FOR Y = 24 TO 27:
       SET(46,Y):
       NEXT Y:
      FOR Y = 24 TO 27:
       SET(47,Y):
       NEXT Y
1060  FOR X = 48 TO 81:
       SET(X,26):
       NEXT X:
      FOR X = 48 TO 81:
       SET(X,27):
       NEXT X
1065  FOR Y = 27 TO 24 STEP - 1:
       SET(82,Y):
       NEXT Y:
      FOR Y = 27 TO 24 STEP - 1:
       SET(83,Y):
       NEXT Y:
      FOR Y = 27 TO 24 STEP - 1:
       SET(84,Y):
       NEXT Y:
      FOR Y = 27 TO 24 STEP - 1:
       SET(85,Y):
       NEXT Y
1068  NEXT T
1069 PRINT @849,"YOU GOT MOST OF THEM RIGHT!!
1070 RETURN
1071 FOR Z = 1 TO 1000:
      NEXT Z:
     CLS
1999 END
2000 FOR X = 28 TO 99:
      SET(X,6):
      NEXT X:
     FOR Y = 6 TO 35:
      SET(98,Y):
      NEXT Y:
     FOR Y = 6 TO 35:
      SET(99,Y):
      NEXT Y:
     FOR X = 99 TO 28 STEP - 1:
      SET(X,35):
      NEXT X
2010 FOR Y = 35 TO 6 STEP - 1:
      SET(29,Y):
      NEXT Y:
     FOR Y = 35 TO 6 STEP - 1:
      SET(28,Y):
```

Program continued

```
      NEXT Y:
     FOR X = 44 TO 49:
      SET(X,12):
      NEXT X:
     FOR X = 44 TO 49:
      SET(X,13):
      NEXT X:
     FOR X = 44 TO 49:
      SET(X,14):
      NEXT X
2020 FOR X = 78 TO 83:
      SET(X,12):
      NEXT X:
     FOR X = 78 TO 83:
      SET(X,13):
      NEXT X:
     FOR X = 78 TO 83:
      SET(X,14):
      NEXT X:
     FOR X = 62 TO 65:
      SET(X,18):
      NEXT X:
     FOR X = 62 TO 65:
      SET(X,19):
      NEXT X:
     FOR X = 62 TO 65:
      SET(X,20):
      NEXT X
2025 FOR Y = 29 TO 26 STEP - 1:
      SET(44,Y):
      NEXT Y:
     FOR Y = 29 TO 26 STEP - 1:
      SET(45,Y):
      NEXT Y:
     FOR Y = 29 TO 26 STEP - 1:
      SET(46,Y):
      NEXT Y:
     FOR Y = 29 TO 26 STEP - 1:
      SET(47,Y):
      NEXT Y
2030 FOR X = 48 TO 81:
      SET(X,26):
      NEXT X:
     FOR X = 48 TO 81:
      SET(X,27):
      NEXT X:
     FOR Y = 26 TO 29:
      SET(82,Y):
      NEXT Y:
     FOR Y = 26 TO 29:
      SET(83,Y):
      NEXT Y:
     FOR Y = 26 TO 29:
      SET(84,Y):
      NEXT Y:
     FOR Y = 26 TO 29:
      SET(85,Y):
      NEXT Y
2040 SET(80,15):
     SET(81,15):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,15):
     RESET(81,15):
     SET(80,17):
     SET(81,17):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,17):
     RESET(81,17):
```

```
     SET(80,19):
     SET(81,19):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,19):
     RESET(81,19)
2041 SET(80,21):
     SET(81,21):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,21):
     RESET(81,21):
     SET(80,23):
     SET(81,23):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,23):
     RESET(81,23)
2050 SET(80,15):
     SET(81,15):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,15):
     RESET(81,15):
     SET(80,17):
     SET(81,17):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,17):
     RESET(81,17):
     SET(80,19):
     SET(81,19):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,19):
     RESET(81,19)
2051 SET(80,21):
     SET(81,21):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,21):
     RESET(81,21):
     SET(80,23):
     SET(81,23):
     FOR Z = 1 TO 50:
      NEXT Z:
     RESET(80,23):
     RESET(81,23)
2055 PRINT @845,"YOU GOT MOST OF THEM WRONG. BOO HOO!!!"
2056 :
     RETURN
2057 END
```

Super Curve Fit

by William L. Morgan

Curve fitting is a useful tool for scientists, science students, and statisticians. It is a method for calculating the closeness to which data conforms to a specified functional curve, such as a logarithmic or parabolic curve. The Super Curve Fit program (see Program Listing) allows you to use the TRS-80 for curve fitting.

Problem One

Many programmers try to solve a specific problem rather than looking for a general solution. Least squares is a method of fitting a curve to a set of points so that the sum of the squares of the distances of the points from the curve is at a minimum. The following equations are used:

$$\begin{aligned}
\Sigma Y_i &= a_0 n + a_1 \Sigma x_i + a_2 \Sigma x_i^2 + \ldots + a_n \Sigma x_i^n \\
\Sigma X_i Y_i &= a_0 \Sigma x_i + a_1 \Sigma x_i^2 + a_2 \Sigma x^3 + \ldots + a_n \Sigma x_i^{n+1} \\
\Sigma X_i^2 Y &= a_0 \Sigma x_i + a_1 \Sigma x_i^3 + a_2 \Sigma x_i^4 + \ldots + a_n \Sigma x_i^{n+2} \\
&\vdots \\
\Sigma X_i^n Y_i &= a_0 \Sigma x_i^n + a_1 \Sigma_i^{n+1} + a_2 \Sigma_i^{n+2} + \ldots + a_n \Sigma x_i^{2n}
\end{aligned}$$

If you replace x_i, x_i^2, x_i^3, and x_i^n with $f_1(x_i)$, $f_2(x_i)$, $f_3(x_i)$, and $f_n(x_i)$, which is a way of saying you perform a function with x, you have:

$$\begin{aligned}
\Sigma Y_i &= a_0 n + a_1 \Sigma f_1(x_i) + a_2 \Sigma f_2(x_i) + \ldots + a_n \Sigma f_n(x_i) \\
\Sigma Y_i f_1(x_i) &= a_0 \Sigma f_1(x_i) + a_1 \Sigma f_1^2(x_i) + a_2 \Sigma f_2(x_i) f_1(x_i) + \ldots + a_n \Sigma f_n(x_i) f_1(x_i) \\
\Sigma Y_1 f_2(x_i) &= a_0 \Sigma f_2(x_i) + a_1 \Sigma f_1(x_i) f_2(x_i) + a_2 \Sigma f_2^2(x_i) + \ldots + a_n \Sigma f_n(x_i) f_2(x_i) \\
&\vdots \\
\Sigma Y_i f_n(x_i) &= a_0 \Sigma f_n(x_i) + a_1 \Sigma f_1(x_i) f_n(x_i) + a_2 \Sigma f_2(x_i) f_n(x_i) + \ldots + a_n \Sigma f_n^2(x_i)
\end{aligned}$$

Using matrix notation, you have:

$$\begin{vmatrix} \Sigma Y_i \\ \Sigma Y_i f_1(x_i) \\ \Sigma Y_i f_2(x_i) \\ \vdots \\ \Sigma Y_i f_n(x_i) \end{vmatrix} = \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{vmatrix} \bullet \begin{vmatrix} n & \Sigma f_1(x_i) & \Sigma f_2(x_i) & \cdots & \Sigma f_n(x_i) \\ \Sigma f_1(x_i) & \Sigma f_1^2(x_i) & \Sigma f_2(x_i) f_1(x_i) & \cdots & \Sigma f_n(x_i) f_1(x_i) \\ \Sigma f_2(x_i) & \Sigma f_1(x_i) f_2(x_i) & \Sigma f_2^2(x_i) & \cdots & \Sigma f_n(x_i) f_2(x_i) \\ \vdots & \vdots & \vdots & & \vdots \\ \Sigma f_n(x_i) & \Sigma f_1(x_i) f_n(x_i) & \Sigma f_2(x_i) f_n(x_i) & \cdots & \Sigma f_n^2(x_i) \end{vmatrix}$$

If you look closely at the X and Y matrices, you will see: for Y, $\sum_{j=1}^{n} Y_i f_j(x_i)$ and for X, $\sum_{k,j=1}^{n} f_j(x_i) f_k(x_i)$. I realized that these are just the cross products of the function list. If I give the computer a function list, it should be able to find the value of a for each function using inversion of the X matrix and multiplication of the Y matrix. That's simple, but how do I know if one list of functions is better than another?

Problem Two

There is a problem when you use least squares, as illustrated in Figure 1. Two functions are shown: $y = 2$ and the square wave function, $f(x)$. Least squares for point P_1 for the square wave function is $(4-0)^2$, or 16; for the $y = 2$ function, least squares is $(4-2)^2$, or 4. Using least squares as a standard, $y = 2$ is a better fit for P_1 than the square wave function. You don't agree; neither do I. Another method must be found. In this example, it would be nice if the x and y could be interchanged, but with some functions, finding the inverse could take forever, or possibly not even be defined at some points.

An alternative to least squares uses normal distances. This involves finding a formula by using the first derivative, however, the TRS-80 can perform differentiation by calculation, so it doesn't need the first derivative. I can make the computer calculate normal distances by moving along the curve until it finds the shortest distance on the interval. This is most of the theory behind Super Curve Fit.

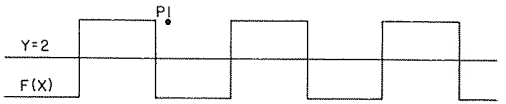


Figure 1. *Using least squares for point P_1. The two functions shown are $y = 2$ and the square wave function, $f(x)$.*

Program Operation

Program lines 10–80 contain the input routine, and lines 90–120 contain

the input correction routine. See Table 1 for a complete outline of the program. Table 2 lists the variables used in Super Curve Fit. Lines 130–160 set up the cross products for the matrices and arrays; lines 200–350 find the inverse of the cross product matrix. Line 400 finds the coefficients for the function list, and lines 410–420 print out the coefficients of the functions. The sum of the normal distances squared is calculated in lines 500–700 and printed out point by point in a list showing experimental x,y, calculated x,y, and the square of the distance between the experimental and the calculated points. Finally, the total sum of the square of the normal distances is printed out.

Lines 1000–1090 make up the function list. The functions must be entered after the CLOAD and before RUN. If you want function 3 to be $f_3(x_i)$ = COS (X + SINX), you would enter this as: 1020 X = COS(X + SIN(X)):RETURN. The routine to calculate f(x) uses ON I GOSUB 1000,1010,. . . ,1090, so you have a maximum of 10 line numbers to define the function. If you need more space, use a GOTO, then hit RETURN when finished. You don't have to use all 10 functions, but you must have one operation for each function you will use, and you can have more that are not used. Run the program, entering the information as the program asks for it. Occasionally, you will have a singular matrix, and no solution is possible. If this happens, a new function list is called for. Next press ENTER, and a point by point summary will be printed on the screen, pausing after 10 lines. Pressing ENTER will restart the summary. At the end of the program the sum of the normal distances will be printed on the screen. Figure 2 shows the results as they would appear on the screen for three sample runs of Super Curve Fit.

This program can handle any type of mathematical function. Because a computer has a limited amount of memory, the number of functions is limited to 10. The program as written uses about 10K of memory; so you may be able to modify the program to handle 12 functions before memory

10–80	Input routine for X_i, Y_i, NF, ND
90–120	Input correction routine for X_i, Y_i
130–160	Routine sets up XM, YC, and clears XI
200–350	Matrix inversion routine
400–420	Finds function coefficients, YK
500–580	Finds the nearest calculated point to the experimental point and the sum of the normal distance squared
590–620	Print routine
700	Subroutine to find value of CY
1000–1090	Function list. The functions must be entered before the program is RUN. (Ex. 1020 X = SIN(X)*COS(X):RETURN)

Table 1. *Outline of Super Curve Fit*

becomes a problem in a 16K Level II machine. There is another method you can use if you want the program to handle more than 10 functions. Run the program using 10 functions and their coefficients. Next, use the first 10 functions and their coefficients as one function in the next set of functions. You will get a new coefficient which, multiplied by each of the first coefficients, gives the new coefficients for each of the first 10 functions. It may not be as accurate as, say, a 19 function fit, but it will be close enough for most applications.

	Matrices
XM(10,10)	Matrix for cross product sum of $\Sigma F_j(X_i)F_k(X_i)$
XI(10,10)	Matrix for inversion of XM
	Arrays
X(100)	Input data array for X_i
Y(100)	Input data array for Y_i
A(10)	Single precision value of YK array
YC(10)	Array for cross product sum of $\Sigma Y_i F_j(X_i)$
YK(10)	Array of coefficients for function list
	Real Variables
CX	Calculated value of X_i
CY	Calculated value of Y_i
D2	Normal distance squared between calculated point and experimental point
DD	Sum of normal distances squared
DL	Lowest D2 value on the interval centered on X_i
EX	Experimental X_i
EY	Experimental Y_i
VI	The interval $\pm .10X_i + X_i$
VL	Value of CX when D2 = DL
VY	Value of CY when D2 = DL
V2	Dummy variable for VL
XD	Determinant of XM matrix
XH	High value of X_i
XL	Low value of Y_i
	Integer Variables
H,I,J,K	Counter variables, mainly for matrix and array operations
N	Counter for data points
ND	Number of data points
NF	Number of functions
LC	Line counter
	String Variables
A$	Dummy for input

Table 2. *Program variables*

A(0) = 6.1347; A(1) = 26.9155; A(2) = − 12.8618; D↑2 = 43.8786

I	X(I)	X(C)	Y(I)	Y(C)	D↑2
1	0.000E 00	− 5.550E − 02	0.000E 00	4.601E 00	2.117E 01
2	2.000E − 02	− 3.550E − 02	5.050E 00	5.163E 00	1.585E − 01
3	5.000E − 02	1.055E − 01	1.080E 01	8.831E 00	3.880E 00
4	1.000E − 01	1.555E − 01	1.320E 01	1.001E 01	1.019E 01
5	2.000E − 01	2.555E − 01	1.360E 01	1.217E 01	2.042E 00
6	4.000E − 01	3.626E − 01	1.420E 01	1.420E 01	1.408E − 03
7	6.000E − 01	5.445E − 01	1.550E 01	1.698E 01	2.184E 00
8	8.000E − 01	7.445E − 01	1.780E 01	1.904E 01	1.551E 00
9	9.000E − 01	8.445E − 01	1.940E 01	1.969E 01	8.840E − 02
10	9.500E − 01	1.006E 00	2.080E 01	2.019E 01	3.696E − 01
11	1.000E 00	1.045E 00	2.176E 01	2.022E 01	2.386E 00

$y = 6.135 + 26.92x - 12.86x^2$

A(0) = 3.8981; A(1) = 78.3055; A(2) = − 154.9586; A(3) = 95.7672; D↑2 = 5.32603

I	X(I)	X(C)	Y(I)	Y(C)	D↑2
1	0.000E 00	− 4.555E − 02	0.000E 00	6.822E − 04	2.075E − 03
2	2.000E − 02	1.515E − 02	5.050E 00	5.049E 00	2.424E − 05
3	5.000E − 02	1.055E − 01	1.080E 01	1.055E 01	6.708E − 01
4	1.000E − 01	1.555E − 01	1.320E 01	1.269E 01	2.655E − 01
5	2.000E − 01	1.822E − 01	1.360E 01	1.360E 01	3.170E − 04
6	4.000E − 01	3.445E − 01	1.420E 01	1.640E 01	4.840E 01
7	6.000E − 01	6.555E − 01	1.550E 01	1.562E 01	1.701E − 02
8	8.000E − 01	8.555E − 01	1.780E 01	1.744E 01	1.331E − 01
9	9.000E − 01	9.212E − 01	1.940E 01	1.940E 01	4.511E − 01
10	9.500E − 01	9.558E − 01	2.080E 01	2.080E 01	3.453E − 05
11	1.000E 01	9.762E − 01	2.176E 01	2.176E 01	5.666E − 05

$y = 3.898 + 78.31x - 155.0x^2 + 95.77x^3$

A(0) = 12.7844; A(1) = 7.4606; A(2) = − 11.8466; D↑2 = 4.90428

I	X(I)	X(C)	Y(I)	Y(C)	D↑2
1	0.000E 00	− 3.700E − 03	0.000E 00	3.004E − 04	1.378E − 05
2	2.000E − 02	2.035E − 03	5.050E 00	5.051E 00	4.812E − 07
3	5.000E − 02	7.665E − 02	1.080E 01	1.080E 01	7.119E − 04
4	1.000E − 01	1.442E − 01	1.320E 01	1.320E 01	1.958E − 03
5	2.000E − 01	1.665E − 01	1.360E 01	1.360E 01	1.129E − 03
6	4.000E − 01	3.445E − 01	1.420E 01	1.534E 01	1.208E 00
7	6.000E − 01	5.445E − 01	1.550E 01	1.685E 01	1.816E 00
8	8.000E − 01	7.445E − 01	1.780E 01	1.834E 01	2.933E − 01
9	9.000E − 01	8.870E − 01	1.940E 01	1.940E 01	1.724E − 04
10	9.500E − 01	1.006E 00	2.080E 01	2.029E 01	2.674E − 01
11	1.000E 00	1.056E 00	2.176E 01	2.066E 01	1.215E 00

$y = 12.78 + 7.461x - 11.85e^{\left(\frac{-x}{.05}\right)}$

Figure 2. *Sample runs*

Program Listing. *Super Curve Fit*

Encyclopedia Loader™

```
10 DEFINT H - N:
   DEFDBL X,Y:
   XL = 1E38:
   XH = - 1E38:
   LC = 0:
   IS = 0
20 DIM A(10),XM(10,10),XI(10,10),YC(10),YK(10),X(100),Y(100)
30 INPUT "ENTER THE NUMBER OF FUNCTIONS";NF:
   IF NF > 10
    THEN
     PRINT "MAX. OF 10 FUNCTIONS-REDO":
     GOTO 30
40 INPUT "ENTER THE NUMBER OF DATA POINTS";ND:
   IF ND > 100
    THEN
     PRINT "MAX. OF 100 DATA POINTS-REDO":
     GOTO 40
50 FOR N = 0 TO ND - 1:
    PRINT "DATA POINT ";N + 1;" X,Y";
60  INPUT X(N),Y(N):
    IF X(N) < XL
     THEN
      XL = X(N)
70  IF X(N) > XH
     THEN
      XH = X(N)
80  NEXT N
90 INPUT "DO YOU WISH TO MAKE CHANGES Y/N";A$:
   IF A$ = "N"
    THEN
     GOTO 130 :
    ELSE
     INPUT "ENTER THE NUMBER OF THE DATA POINT   TO BE CHANGED, THE
     N X,Y";N,X,Y:
     X(N - 1) = X:
     Y(N - 1) = Y:
     GOTO 90
100 XL = 1E38:
    XH = - 1E38:
    FOR N = 0 TO ND - 1:
     IF XL > X(N)
      THEN
       XL = X(N)
110  IF XH < X(N)
      THEN
       XH = X(N)
120  NEXT N
130 XM(0,0) = ND:
    FOR I = 0 TO NF:
     YC(I) = 0:
     NEXT I:
    FOR N = 0 TO ND - 1:
     YC(0) = YC(0) + Y(N)
140  FOR I = 1 TO NF:
      X = X(N):
      ON I GOSUB 1000,1010,1020,1030,1040, 1050,1060,1070,1080,1090:
      YC(I) = YC(I) + Y(N) * X:
      XM(0,I) = XM(0,I) + X:
      XM(I,0) = XM(0,I):
      XF = X
150   FOR J = 1 TO NF:
       X = X(N):
       ON J GOSUB 1000,1010,1020,1030,1040, 1050,1060,1070,1080,1090
       :
       XM(I,J) = XM(I,J) + XF * X:
       NEXT J,I,N
160   FOR I = 0 TO NF:
       FOR J = 0 TO NF:
```

Program continued

```
        XI(I,J) = 0:
        NEXT J:
       XI(I,I) = 1:
       NEXT I
200   FOR J = 0 TO NF
210    I = J - 1
220    I = I + 1:
       IF I = NF + 1
        THEN
         GOTO 350
230    IF XM(I,J) < > 0
        THEN
         GOTO 250 :
        ELSE
         IF I = NF AND J = NF
          THEN
           GOTO 240 :
          ELSE
           GOTO 220
240    PRINT "SINGULAR MATRIX,NO SOLUTION":
       END
250    FOR H = 0 TO NF:
        XT = XM(J,H)
260     XM(J,H) = XM(I,H):
        XM(I,H) = XT
270     XT = XI(J,H):
        XI(J,H) = XI(I,H):
        XI(I,H) = XT:
        NEXT H
280    XD = XM(J,J):
       FOR H = 0 TO NF
290     XM(J,H) = XM(J,H) / XD:
        XI(J,H) = XI(J,H) / XD:
        NEXT H
300    FOR H = 0 TO NF:
        IF J = H
         THEN
          GOTO 340
310      XD = XM(H,J):
         FOR K = 0 TO NF
320       XM(H,K) = XM(H,K) - XM(J,K) * XD:
          XI(H,K) = XI(H,K) - XI(J,K) * XD
330       NEXT K
340     NEXT H
350    IF I = < NF
        THEN
         GOTO 220 :
        ELSE
         NEXT J
400   FOR I = 0 TO NF:
       YK(I) = 0:
       FOR J = 0 TO NF:
        YK(I) = YK(I) + YC(J) * XI(I,J):
        NEXT J,I
410    FOR I = 0 TO NF:
        PRINT "A(";I;")= ";YK(I):
        NEXT I
420    INPUT "PRESS ENTER TO GET POINT BY POINT RESULTS";A$
500    PRINT USING "% %";" ";"I";:
       PRINT USING "%         %";" X(I)"; " X(C)";" Y(I)";" Y(C)";" D
        2":
       IF IS = 1
        THEN
         GOTO 600
510    VI = CSNG((XH - XL) / 20):
       FOR I = 0 TO NF:
        A(I) = CSNG(YK(I)):
        NEXT I
520    FOR N = 0 TO ND - 1:
        EX = CSNG(X(N)):
```

```
        EY = CSNG(Y(N)):
        VL = EX - VI
530     DL = 1E38:
        FOR K = 0 TO 20:
         CX = EX - VI + K * VI / 10
540      GOSUB 700:
         D2 = (EX - CX) [ 2 + (EY - CY)   2:
         IF D2 < DL
          THEN
           DL = D2:
           VL = CX:
           VY = CY:
           NEXT K :
         ELSE
          NEXT K:
         DL = 1E38
550    V2 = VL:
       FOR K = 0 TO 20:
        CX = V2 - VI / 10 + K * VI / 100
560     GOSUB 700:
        D2 = (EX - CX) [ 2 + (EY - CY) [ 2:
        IF D2 < DL
         THEN
          DL = D2:
          VL = CX:
          VY = CY:
          NEXT K :
        ELSE
         NEXT K:
        DL = 1E38
570   V2 = VL:
      FOR K = 0 TO 20:
       CX = V2 - VI / 100 + K * VI / 1000
580    GOSUB 700:
       D2 = (EX - CX)   2 + (EY - CY)   2:
       IF D2 < DL
        THEN
         DL = D2:
         VL = CX:
         VY = CY:
         NEXT K :
       ELSE
        NEXT K
590  IF LC = 10
      THEN
       LC = 1:
       INPUT "A PROGRAM HOLD PRESS ENTER TORESTART";A$:
       IS = 1:
       GOTO 500 :
      ELSE
       LC = LC + 1
600   PRINT USING "###!";N + 1;:
      PRINT " ";
610   PRINT USING "##.###     ";EX;VL;EY;VY;DL:
      DD = DD + DL:
      NEXT N
620  PRINT "SUM OF NORMAL DISTANCES SQUARED IS ";DD:
     END
700  CY = A(0):
     FOR I = 1 TO NF:
      X = CX:
      ON I GOSUB 1000,1010,1020,1030, 1040,1050,1060,1070,1080,1090:
      CY = CY + A(I) * CSNG(X):
      NEXT I:
     RETURN
1000 RETURN
1010 RETURN
1020 RETURN
1030 RETURN
1040 RETURN
```

program continued

```
1050 RETURN
1060 RETURN
1070 RETURN
1080 RETURN
1090 RETURN
```

GAMES

Queen Rama's Cave
TRS-80 Jukebox

Queen Rama's Cave

by John Corbani

Adventure programs have been around almost as long as computers have had CRT terminals. If you have ever wondered how they work or if you could write one, read on. Queen Rama's Cave is written as a demonstration program to illustrate how easy it is to write this type of program in BASIC. Most of the program, which fits within 16K, describes the various parts of the cave and the objects and creatures who live there. The basic structure of the program is separate from the descriptions; so you can customize the program to any environment you wish. The cave could just as well be a ship, space station, house, department store, or office building. Your imagination is the only limit.

The starting point for writing an adventure program is to lay out a three-dimensional view of the environment. This program uses a three-dimensional array to identify all points the player can visit. Each dimension in the array contains three elements. This allows three levels, each with 3×3 or nine stopping points. Twenty of the 27 possible points form the adventure's starting point, the cave entrance, and the bowels of the cave as shown in Figure 1. Once you have defined the structure, you can determine the allowable paths from point to point. These are corridors, stairways, and ladders. You now have enough information to write a description of each point and to specify the moves that can be made from that point. The program stores the descriptions in string array DE$(n) and the possible moves in DI$(n).

An adventure needs a goal, and of course there must be some danger to add spice. Overcoming danger requires weapons and/or some fancy talking. The goal is a giant diamond. The diamond, weapons, and other treasures to be found in the cave are identified as B1$ through B5$. Horrible creatures who live in the cave are identified as A$(1) through A$(5). The fancy words are listed in SL$. Using the fancy words to get out of a tight spot brings a response specified in B$(1) through B$(5). There are other words you can use, but you had better be armed or you will wish you had kept your mouth shut. These words are stored in SK$. A random factor should exist; in this case, it is the Greep. He prowls the cave, and anyone who spends too much time underground is sure to run into him. He gives a warning on first contact but vents his wrath on the second meeting.

You are given certain abilities and a genie to accompany you along the way. Key words are GET, DROP, HELP, START, and the abbreviations of

the six possible directions of travel, N, S, E, and W (north, south, east, and west) and U and D (up and down). You and your genie can move from point to point in the cave by specifying one or a combination of two directions of travel. If the specified direction is allowable, your position in the array PO$(n,n,n) is updated. The array variable then points to the point description, possible directions, contents, and so on. This indirect addressing makes program housekeeping and modification a simple chore.

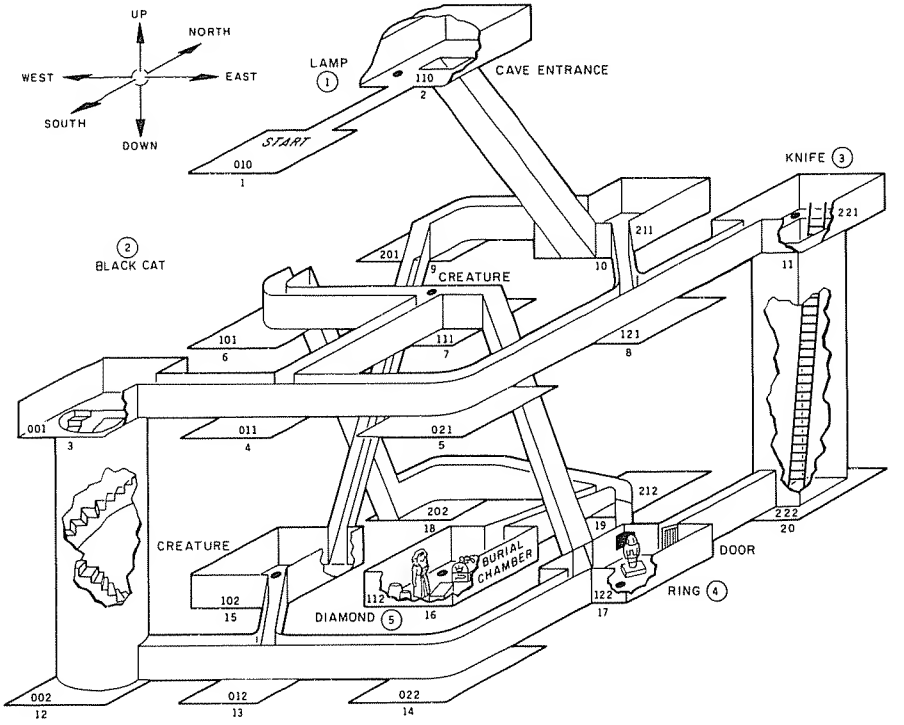


Figure 1. *Diagram of Queen Rama's Cave*

Determining what the player is saying, parsing the sentence, is where the string features of Microsoft BASIC really shine. The program is not very sophisticated but has enough sense to tell the player when it is stumped. Player input starts at line 600 and is gradually digested. Flow jumps to the key subroutines if the first word of the input sentence is a key word. If the input is one or two characters, the program compares the characters to the allowable moves from the present position (DI$(n)). If the move is not allowed, the program gives you a wise remark and looks for new input.

The program checks to see if you ran into the Greep. The variable T contains the number of moves you have made. At the tenth move, the Greep

gives a warning. At the seventeenth move the Greep transports you to a random point in the cave. At the twenty-fifth move, you are stripped of your possessions and thrown from the cave. All this happens in the subroutine that starts at line 470.

After the Greep has been handled, the move is executed at 650 by going through the subroutine at 740 and updating the terms of the three-dimensional array. A description of the new location and of all items there is printed at 1080, and the program jumps back to 650. The program then checks the position for points 7 or 15 to see if the player ran into a creature. The subroutine at 1280 takes care of all of the confrontations. You can talk your way by the creature or commit mayhem on the creature if you are armed. A suitable message is printed depending on the outcome of the encounter. Then the program goes back to line 600.

The key word START is a cop-out to use when you are hopelessly lost. It just starts the program over again. The key word HELP prints all of the commands on the screen and lists all of the objects the player is carrying. The n = 0 element of B1$(n) through B5$(n) indicates that an object belongs to the player. Other values of n indicate the place where the object is resting.

The key word GET causes a jump to 1160 where the player's objects are counted. If you already have three, you can't pick up another one without dropping one. Detour to 1420 and then back. Now there is a jump to 1600 where the program determines the last word in the input sentence, then back to 1180 to see if it is one of the portable objects. If it is, and if the object the player wants is in the room, it is given to him by making its array element equal 0. If not, a message is printed outlining the problem. The program finally loops back to 600 for more input.

The key word DROP causes a jump to 1440. The last word in the sentence is obtained in 1600 and checked against the names of all objects at 1450. If the player has the object, the program drops it by changing its array number to the number of the player's position.

A list of all variables is printed at the end of the program. This is invaluable when you are writing or debugging, but you can drop it from the final program to save memory space. The form of the program adds a lot to the readability and understandability of the listing. I have taken full advantage of multi-line statements and have added spaces freely for line formatting. All lines contain fewer than 64 characters, and I added a line feed (down arrow) at the end of all upper lines of a multi-line statement. Text that will be printed on the screen is not indented so the formatting is visualized easily.

The simple approach to program formatting in which every statement has its own line works fine when the problems are small, the programmer is a beginner, and the language is inflexible. Using that approach on this program would result in a program that would require 32K to run. As your skill

builds and your knowledge of BASIC increases, use every tool that Microsoft provides. The form of a sentence or paragraph varies to suit the subject matter; not the other way around. Program the same way.

The program is a lot of fun for kids and is not a bad demonstration of the computer to the uninitiated. Leave the computer on with the illustration nearby, and most people will start to visualize in three dimensions.

Program Listing. *Queen Rama's Cave*

Please note: This program listing has not been formatted to preserve the author's spacing.

```
10 '                QUEEN  RAMA'S  CAVE              11/5/81
                       BY
                   JOHN CORBANI
                   2455 CALLE LINARES
                   SANTA BARBARA CA, 93109
20 '                 "Q" TAPE #6, 5-140

30 CLEAR 500: DIM A$(5), B$(5), D$(5), PO(3,3,3), DE$(20),
   DI$(20), B1$(20),  B2$(20), B3$(20), B4$(20), B5$(20)
40 D$(1)="YOU CANT GO THAT WAY":
   D$(2)="NOT THAT WAY TODAY":
   D$(3)="NO WAY":
   D$(4)="NO! NOT THIS TIME":
   D$(5)="THAT WAY WILL KILL YOU"
50 A$(1)="SERPENT":
   A$(2)="SNIVELOPOGOUS":
   A$(3)="DRAGON":
   A$(4)="BERGRUMP":
   A$(5)="BUSHWAUGH"
60 B$(1)=
"HE DECIDED TO LET YOU STAY FOR A MINUTE. BE QUICK BEFORE HE
CHANGES HIS MIND.":
   B$(2)=
"IF YOU HURRY UP, YOU CAN SNEAK BY HIM.":
   B$(3)=
"HE APPEARS TO BE ASLEEP. MAYBE YOU CAN GET BY."
70 B$(4)=
"SOME CREATURES ARE DUMB. TRY TO GO AROUND HIM":
    B$(5)=
"HE IS LOOKING THE OTHER WAY RIGHT NOW"
80 B1$="A SHINY BRASS LAMP":
   B2$="A VERY BLACK CAT":
   B3$="A SHARP KNIFE":
   B4$="A SMALL GOLD RING":
   B5$="A GIANT DIAMOND"
90 B1$(2)=B1$: B2$(RND(16)+4)=B2$: B3$(11)=B3$: B4$(17)=B4$:
   B5$(16)=B5$
100 O1$="          THERE IS ": O2$=" ON THE FLOOR":
   H2$="          YOU HAVE "
110 SK$="KILLMAIMDROWNHITSTABMURDERGETSMASH":
   SL$="LOVEPATSNUGGLEFEEDCALLPETRUB"
120 DI$(1)="N-":          DI$(2)="S-NDN":
    DI$(3)="E-D-":        DI$(4)="N-E-W-":
    DI$(5)="N-W-":        DI$(6)="E-NDN":
    DI$(7)="S-W-EDE":     DI$(8)="N-S-NWN":
    DI$(9)="E-SDS"
130 DI$(10)="W-USU.SES":  DI$(11)="S-D-":
    DI$(12)="E-U-":       DI$(13)="E-W-NWN":
    DI$(14)="N-W-":       DI$(15)="NUN.SES":
    DI$(16)="N-":         DI$(17)="S-NWN.UWU"
140 DI$(18)="E-USU":      DI$(19)="S-W-ESE":
    DI$(20)="S-U-"
150 PO(0,1,0)=1:      PO(1,1,0)=2:      PO(0,0,1)=3:
    PO(0,1,1)=4:      PO(0,2,1)=5:      PO(1,0,1)=6:
    PO(1,1,1)=7:      PO(1,2,1)=8:      PO(2,0,1)=9
160 PO(2,1,1)=10:     PO(2,2,1)=11:     PO(0,0,2)=12:
    PO(0,1,2)=13:     PO(0,2,2)=14:     PO(1,0,2)=15:
    PO(1,1,2)=16:     PO(1,2,2)=17:     PO(2,0,2)=18:
    PO(2,1,2)=19:     PO(2,2,2)=20
170  DE$(1)=
"    YOU ARE TWENTY YARDS SOUTH OF A LOW CAVE AT THE BASE OF
A TALL CLIFF."
180 DE$(2)=CHR$(28)+CHR$(31)+
"YOU ARE IN THE MOUTH OF A CAVE. A STONE STAIRWAY LEADS DOWN
TO THE NORTH."
190 DE$(3)=
"THIS ROOM HAS A HIGH DOMED CEILING. A LOW DOORWAY IS IN THE
```

Encyclopedia Loader

Program continued

```
EAST WALL. A STAIRWAY WINDS DOWN THE SIDE OF A ROUND HOLE IN
THE FLOOR."
200 DE$(4)=
"THE ROOF IS VERY LOW NOW AND YOU MUST BEND OVER TO CONTINUE.
YOU ARE IN AN EAST WEST PASSAGE THAT HAS AN OPENING  TO THE
NORTH."
210 DE$(5)=
"YOU ARE IN A TUNNEL THAT CURVES FROM THE WEST TO THE NORTH."
220 DE$(6)=
"THERE IS A SMALL WATERFALL TRICKLING DOWN THE SOUTH WALL OF
THE TUNNEL. A NARROW CORRIDOR DESCENDS INTO THE DARKNESS TO
THE NORTH. YOU CAN JUST SEE A WIDE CRACK TO THE EAST."
230 DE$(7)=
"THIS CORRIDOR COMES FROM THE WEST AND PITCHES STEEPLY
DOWNWARD TO THE EAST. ANOTHER PASSAGEWAY CURVES TO THE SOUTH."
240 DE$(8)=
"YOU ARE AT A POINT WHERE A NORTH SOUTH PATH BRANCHES TO THE
NORTH WEST."
250 DE$(9)=
"THE TUNNEL WIDENS AS IT CURVES FROM THE EAST TO THE TOP OF A
MARBLE STAIRWAY LEADING DOWN TO THE SOUTH."
260 DE$(10)=
"YOU ARE IN A TRIANGULAR CHAMBER AT THE BOTTOM OF A STONE
STAIRWAY RISING TO THE SOUTH. A DOORWAY OPENS WEST AND A ROUGH
HOLE LEADS SOUTHEAST."
270 DE$(11)=
"YOU ARE AT THE TOP OF A VERTICAL SHAFT DROPPING AWAY INTO THE
GLOOM. AN OPENING IN THE WALL LEADS SOUTH. YOU CAN JUST SEE
THE TOP OF A LADDER AT THE EDGE OF THE SHAFT."
280 DE$(12)=
"YOU ARE AT THE BOTTOM OF A VERY TALL ROUND SHAFT RISING INTO
THE GLOOM. YOU ARE STANDING ON A PATH LEADING FROM AN OPENING
IN THE EAST WALL TO THE FOOT OF A STAIRWAY CURVING UP INTO THE
DARK."
290 DE$(13)=
"AT THIS POINT THE EAST WEST TUNNEL SHOWS A SMALL OPENING GOING
TO THE NORTHWEST."
300 DE$(14)=
"YOUR TUNNEL CURVES AS IT GOES FROM THE NORTH TO THE WEST."
310 DE$(15)=
"YOU ARE IN A LARGE CHAMBER. A MARBLE STAIRWAY COMES DOWN FROM
THE  NORTH. A TUNNEL CURVES AWAY TO THE SOUTHEAST."
320 DE$(16)=
"THIS IS THE TREASURE ROOM OF QUEEN RAMA. ANCIENT TAPESTRIES
FRAME THE DOORWAY IN THE NORTH WALL. A STATUE OF THE QUEEN SITS
ON A GOLD THRONE AGAINST THE SOUTH WALL. CHESTS OF GOLD AND
JEWELS ARE PLACED TO EITHER SIDE."
330 DE$(17)=
"YOU HAVE ENTERED A GREAT HALL WHERE A STONE GOD STANDS IN THE
MIDDLE OF THE FLOOR. HE IS FACING A RISING CORRIDOR IN THE WEST
WALL. PASSAGES OPEN TO THE SOUTH AND NORTHWEST. A DOOR WITH NO
HANDLE JUST CLOSED IN THE NORTH WALL."
340 DE$(18)=
"A NARROW CORRIDOR RISES STEEPLY TO THE SOUTH AND AN OPENING
CURVES TO THE EAST."
350 DE$(19)=
"THE ROOF OF THE CAVERN IS NOW ONLY TWO FEET HIGH. AS YOU CRAWL
ALONG; OPENINGS TO THE WEST, SOUTH AND SOUTH EAST ARE VISIBLE."
360 DE$(20)=
"YOU ARE NOW AT THE BOTTOM OF A VERTICAL SHAFT. A LADDER RISES
INTO THE DARKNESS. AN OPENING TO THE SOUTH IS NEARBY."
370 S1$=CHR$(13)+
"    YOU ARE AT THE ENTRANCE OF THE LONG LOST TREASURE CAVE OF
QUEEN RAMA THE GREAT.  IT HAS TAKEN":
    S2$=                                "MOVES TO GET HERE. I
WILL ACCOMPANY YOU IF YOU WISH TO EXPLORE THE CAVE."
380 S3$=                                        " TRY TO FIND
THE QUEEN'S GREAT DIAMOND AND BRING IT BACK TO THIS POINT.

                  WATCH OUT FOR THE GREEP!
```

```
"
390 '
    START PROGRAM

400 CLS: PRINT
"           THE TREASURE CAVE OF QUEEN RAMA": PRINT: PRINT
"  DURING THIS EXPEDITION YOU ARE ALLOWED TO MOVE NORTH (N),
SOUTH (S), EAST (E), WEST (W), UP (U), DOWN (D) OR A COMBI-"
410 PRINT
"NATION OF ANY TWO. AFTER MAKING YOUR SELECTION, PRESS (ENTER)"
420 PRINT
"  IF YOU WISH TO PICK SOMETHING UP, TYPE (GET) AND THE
OBJECT'S NAME. IF YOU WISH TO PUT SOMETHING DOWN, TYPE (DROP)"
430 PRINT
"AND THE OBJECT'S NAME. IF YOU FORGET WHAT YOU HAVE, TYPE
(HELP) AND I, THE GENIE WILL HELP. (START) RESTARTS THE GAME."
440 PRINT: PRINT
"  YOU ARE FACING NORTH AND A CAVE IS JUST VISIBLE AT THE
BASE OF A CLIFF IN FRONT OF YOU. AS YOU LOOK AROUND YOU SEE
NOTHING BUT ROLLING DUNES. YOU MUST CHOOSE A DIRECTION.":
PRINT
450 EW=1: ST=1: GOTO 600
460 '
    GREEPS WARNING

470 CLS: FOR A%=1 TO 100: PRINT @ RND(1000), "*";: NEXT:
    PRINT @ 145, "THE GREEP IS IN FRONT OF YOU.": FOR A=1
    TO 800: NEXT
480 IF T=17 THEN PRINT
"THE GREEP HAS TRANSPORTED YOU SOMEWHERE ELSE IN THE CAVE.
CONSIDER YOURSELF LUCKY TO BE ALIVE.":
NS=RND(3)-1: EW=RND(3)-1: UD=RND(2): I$="-"
490 IF T>24 THEN GOSUB 520
500 RETURN
510 '
    GREEPS WRATH

520 PRINT
"                THIS IS THE FINAL TIME     DUCK !": FOR A=1
TO 1000: NEXT: FOR A=1 TO 6: PRINT STRING$(192,191): CLS:
NEXT: PRINT
"YOU ARE LYING IN A BLOODY HEAP IN THE MIDDLE OF THE DESERT.
THE GREEP HAS STUFFED YOU STRAIGHT UP THROUGH THE ROCKS."
530 IF B1$(0)>"" THEN B1$(2)=B1$: B1$(0)=""
540 IF B2$(0)>"" THEN B2$(RN)=B2$: B2$(0)=""
550 IF B3$(0)>"" THEN B3$(RN)=B3$: B3$(0)=""
560 IF B4$(0)>"" THEN B4$(RN)=B4$: B4$(0)=""
570 IF B5$(0)>"" THEN B5$(RN)=B5$: B5$(0)=""
580 NS=0: EW=1: UD=0: I$="-": P=0: S=0: T=0: RETURN
590 '
    INPUT ROUTINE
600 INPUT "WHAT IS YOUR WISH "; I$:
    IF I$="START" THEN RUN ELSE
    IF I$="HELP" THEN GOSUB 720: GOTO 600ELSE
    IF LEFT$(I$,3)="GET" THEN GOSUB 1160: GOTO 600ELSE
    IF LEFT$(I$,4)="DROP" THEN GOSUB 1440: GOTO 600
610 IF LEN(I$)=1 THEN I$=I$+"-" ELSE IF LEN(I$)>2 THEN PRINT
"I DONT UNDERSTAND YOU": GOTO 600
620 RN=PO(NS,EW,UD): FOR A=1 TO 10:
    IF I$=MID$(DI$(RN),A,2) THEN A=20
630 NEXT: IF A<15 THEN PRINT D$(RND(5)): GOTO 600
640 PRINT: T=T+1: S=S+1: IF T=10 OR T=17 OR T>24 THEN
    GOSUB 470
650 GOSUB 740: GOSUB 1080:
    IF RN=15 OR RN=7 THEN A$=A$(RND(5)): GOSUB 1280ELSE
    IF RN=2 THEN GOSUB 690
660 IF RN=8 OR RN=9 THEN GOSUB 1050ELSE
    IF RN=1 THEN ST=1
670 GOTO 600
680 '
```

Program continued

```
     ENTRANCE TO THE CAVE

690 PRINT S1$; S; S2$ S3$: S=0: T=0: GOSUB 1530:
     IF ST=1 THEN ST=0
700 RETURN
710 '
     HELP ROUTINE

720 PRINT: PRINT
"THE FOLLOWING KEYS OR WORDS ARE VERY USEFUL. N S  E W  U D
GET   DROP    HELP    START    (ENTER). THERE ARE MORE BUT
YOU WILL HAVE TO FIND THEM YOURSELF.":
     PRINT: GOSUB 1530: RETURN
730 '
     MOVE TO NEW POSITION
.
740 IF I$="N-" THEN NS=NS+1: RETURN
750 IF I$="NE" THEN NS=NS+1: EW=EW+1: RETURN
760 IF I$="NW" THEN NS=NS+1: EW=EW-1: RETURN
770 IF I$="NU" THEN NS=NS+1: UD=UD-1: RETURN
780 IF I$="ND" THEN NS=NS+1: UD=UD+1: RETURN
790 IF I$="S-" THEN NS=NS-1: RETURN
800 IF I$="SE" THEN NS=NS-1: EW=EW+1: RETURN
810 IF I$="SW" THEN NS=NS-1: EW=EW-1: RETURN
820 IF I$="SU" THEN NS=NS-1: UD=UD-1: RETURN
830 IF I$="SD" THEN NS=NS-1: UD=UD+1: RETURN
840 IF I$="E-" THEN EW=EW+1: RETURN
850 IF I$="EN" THEN EW=EW+1: NS=NS+1: RETURN
860 IF I$="ES" THEN EW=EW+1: NS=NS-1: RETURN
870 IF I$="EU" THEN EW=EW+1: UD=UD-1: RETURN
880 IF I$="ED" THEN EW=EW+1: UD=UD+1: RETURN
890 IF I$="W-" THEN EW=EW-1: RETURN
900 IF I$="WN" THEN EW=EW-1: NS=NS+1: RETURN
910 IF I$="WS" THEN EW=EW-1: NS=NS-1: RETURN
920 IF I$="WU" THEN EW=EW-1: UD=UD-1: RETURN
930 IF I$="WD" THEN EW=EW-1: UD=UD+1: RETURN
940 IF I$="U-" THEN UD=UD-1: RETURN
950 IF I$="UN" THEN UD=UD-1: NS=NS+1: RETURN
960 IF I$="US" THEN UD=UD-1: NS=NS-1: RETURN
970 IF I$="UE" THEN UD=UD-1: EW=EW+1: RETURN
980 IF I$="UW" THEN UD=UD-1: EW=EW-1: RETURN
990 IF I$="D-" THEN UD=UD+1: RETURN
1000 IF I$="DN" THEN UD=UD+1: NS=NS+1: RETURN
1010 IF I$="DS" THEN UD=UD+1: NS=NS-1: RETURN
1020 IF I$="DE" THEN UD=UD+1: EW=EW+1: RETURN
1030 IF I$="DW" THEN UD=UD+1: EW=EW-1: RETURN
1040 RETURN
1050 IF B1$(0)>"" THEN RETURN ELSE PRINT
"IT IS VERY DANGEROUS TO CONTINUE WITHOUT SOME LIGHT":
     IF T<14 THEN  T=14
1060 RETURN
1070 '
     DETERMINE ROOM NUMBER,
     PRINT DESCRIPTION AND CONTENTS

1080 RN=PO(NS,EW,UD): PRINT DE$(RN)
1090 IF B1$(RN)>"" THEN PRINT O1$ B1$ O2$
1100 IF B2$(RN)>"" THEN PRINT O1$ B2$ O2$
1110 IF B3$(RN)>"" THEN PRINT O1$ B3$ O2$
1120 IF B4$(RN)>"" THEN PRINT O1$ B4$ O2$
1130 IF B5$(RN)>"" THEN PRINT O1$ B5$ O2$
1140 RETURN
1150 '
     GET OBJECTS

1160 IF P>2 THEN GOSUB 1420: GOTO 1090
1170 GOSUB 1600
1180 FOR A=7 TO 20:
     IF MID$(B1$(RN),A,LI)=I$ THEN B1$(0)=B1$: B1$(RN)="":
     GOTO 1250
```

```
1190 IF MID$(B2$(RN),A,LI)=I$ THEN B2$(0)=B2$: B2$(RN)="":
     GOTO 1250
1200 IF MID$(B3$(RN),A,LI)=I$ THEN B3$(0)=B3$: B3$(RN)="":
     GOTO 1250
1210 IF MID$(B4$(RN),A,LI)=I$ THEN B4$(0)=B4$: B4$(RN)="":
      GOTO 1250
1220 IF MID$(B5$(RN),A,LI)=I$ THEN B5$(0)=B5$: B5$(RN)="":
      GOTO 1250
1230 NEXT: IF A<30 THEN PRINT "I CANT GET THAT HERE."
     ELSE PRINT "YOU HAVE THE " I$
1240 GOTO 1090
1250 P=P+1: A=35: GOTO 1230
1260 RETURN
1270 '
     HANDLE CREATURES

1280 PRINT
     "A " A$ " IS BLOCKING YOUR WAY. WHAT DO YOU WANT TO DO ":
     INPUT I$: LI=LEN(I$): FOR A=1 TO 10: IF MID$(I$,A,1)=" "
     THEN I$=LEFT$(I$,A-1): LI=LEN(I$): A=20
1290 NEXT: IF B2$(0)>"" OR B3$(0)>"" THEN W=1 ELSE W=0
1300 FOR A=1 TO 40: T$=MID$(SK$,A,LI): IF T$=I$ THEN A=50: N=1:
     GOTO 1320
1310 T$=MID$(SL$,A,LI): IF T$=I$ THEN A=50: N=2
1320 NEXT: IF A<45 THEN PRINT
"I CANT UNDERSTAND YOU AND IT'S GETTING HOT IN HERE.":
     GOTO 1280
1330 IF W>0 AND N=1 THEN 1340ELSE ON N GOTO 1350,1390
1340 PRINT "THE " A$ " JUST DISSAPEARED IN A CLOUD OF SMOKE.":
     TS=TS+100: RETURN
1350 CLS: PRINT
"YOU SHOULDN'T HAVE FOOLED AROUND WITH THE " A$ ". YOU ARE
GOING TO BE E  E   E                    ";: FOR A=1 TO 1500:
NEXT: PRINT "THROWN OUT"
1360 IF B4$(0)>"" THEN B4$(RN)=B4$: B4$(0)=""
1370 IF B5$(0)>"" THEN B5$(RN)=B5$: B5$(0)=""
1380 B1$(0)="": B1$(2)=B1$: NS=0: EW=1: UD=0: P=0: GOSUB 1080:
     RETURN
1390 PRINT B$(RND(5)): IF B2$(0)>"" THEN B2$(RN)=B2$: B2$(0)=""
1400 RETURN
1410 '
     TOO MANY OBJECTS

1420 GOSUB 1530: INPUT
"YOU CAN CARRY ONLY THREE THINGS AT A TIME. YOU WILL HAVE TO
GIVE SOMETHING UP. WHAT WILL IT BE "; I$
1430 '
     DROP OBJECTS

1440 GOSUB 1600
1450 FOR A=8 TO 20:
     IF MID$(B1$(0),A,LI)=I$ THEN B1$(RN)=B1$: B1$(0)="":
     GOTO 1510
1460 IF MID$(B2$(0),A,LI)=I$ THEN B2$(RN)=B2$: B2$(0)="":
     GOTO 1510
1470 IF MID$(B3$(0),A,LI)=I$ THEN B3$(RN)=B3$: B3$(0)="":
     GOTO 1510
1480 IF MID$(B4$(0),A,LI)=I$ THEN B4$(RN)=B4$: B4$(0)="":
     GOTO 1510
1490 IF MID$(B5$(0),A,LI)=I$ THEN B5$(RN)=B5$: B5$(0)="":
     GOTO 1510
1500 NEXT: IF A<30 THEN PRINT "YOU DONT HAVE A " I$:
     RETURN ELSE PRINT "THE " I$ " IS ON THE FLOOR": RETURN
1510 P=P-1: A=40: GOTO 1500
1520 '
      PRINT OBJECTS CARRIED

1530 IF P=0 THEN PRINT "YOUR HANDS ARE EMPTY.": ELSE
     IF B1$(0)>"" THEN PRINT H2$ B1$(0)
1540 IF B2$(0)>"" THEN PRINT H2$ B2$(0)
1550 IF B3$(0)>"" THEN PRINT H2$ B3$(0)
```

Program continued

```
1560 IF B4$(0)>"" THEN PRINT H2$ B4$(0)
1570 IF B5$(0)>"" THEN PRINT H2$ B5$(0)
1580 RETURN
1590 '
     GET LAST WORD OF I$

1600 LI=LEN(I$): FOR A=LI TO 1 STEP -1: IF MID$(I$,A,1)=" "
     THEN I$=MID$(I$,A+1,20): LI=LEN(I$): A=0
1610 NEXT: RETURN
1620 '
     LIST OF VARIABLES

1630 'A        = LOOP VARIABLE
1640 'A%       = FAST LOOP VARIABLE
1650 'A$       = TEMPORARY STRING VARIABLE
1660 'A$(1)
     A$(5)    = NAMES OF CREATURES TO BE MET IN CAVE
1670 'B$(1)
     B$(5)    = CREATURES' STATUS
1680 'B1$
     B5$      = OBJECT DESCRIPTIONS
1690 'B1$(0)
     B5$(0)   = OBJECTS HELD BY PLAYER
1700 'B1$(1)
     B5$(20)  = "MAILBOXES" FOR B1$ THROUGH B5$ AT ROOMS 1-20
1710 'D$(1)
     D$(5)    = ERROR MESSAGES. (DIRECTION)
1720 'DE$(1)
     DE$(20)  = DESCRIPTION OF ROOMS 1-20
1730 'D1$(1)
     DI$(20)  = POSSIBLE MOVE DIRECTIONS FROM PRESENT POSITION.
1740 'EW       = EAST WEST VARIABLE  (0-2)
1750 'H2$      = LEADER STRING FOR OBJECTS HELD BY PLAYER.
1760 'I$       = INPUT STRING
     LI       = LENGTH OF I$
1770 'NS       = NORTH SOUTH VARIABLE  (0-2)
1780 'O1$
     O2$      = LEADER AND TRAILER STRINGS FOR OBJECTS AT
                POSITIONS.
1790 'PO(NS,EW,UD)= POSITION (ROOM) NUMBER  (RN)  (0-20)
1800 'RN       = ROOM NUMBER
1810 'S        = TOTAL NUMBER OF VALID MOVES.
     ST       = START.  1 AT POSITION 1. 0 AT POSITION 2
1820 'SL$      = LOVING WORDS TO CREATURE
1830 'SK$      = KILLING WORDS TO CREATURE
1840 'S1$
     S2$
     S3$      = MESSAGE AT OPENING OF CAVE
1850 'T        = NUMBER OF MOVES. (MODIFIED BY GREEP, LAMP)
1860 'UD       = UP DOWN VARIABLE  (0-2)
     W        = WEAPONS.    0=NO     1=YES
```

TRS-80 Jukebox

by Craig A. Lindley

A back issue of *80 Microcomputing* had the phrase "All work and no play. . ." printed on the front cover. After reading that issue from cover to cover I decided that I had fallen into that dull category. After thinking a while about writing more exciting programs, I came up with the idea of a computerized jukebox. The idea seemed challenging because it would require both BASIC and assembly-language programming along with a little music theory. I decided that I didn't want to use any external hardware to produce the music. I wanted to do it all through the single-bit cassette port. I also wanted to be able to change easily the songs that the jukebox could play. The program shown here is the result of my efforts.

Making Sounds with One Bit

I pulled out all my old computer magazines and looked for articles on the generation of sound by a computer. All I could find were programs for producing sounds for gun shots, race cars, tanks, and so on. You name a noise, and there is an article somewhere on how to generate it. Music generation, on the other hand, has had limited publication, and most of the articles I found in my library were esoteric music production programs using fast Fourier transforms and such, usually with some specialized hardware involved. I felt that the jukebox should be a monotonic instrument, that is, one that plays one note at a time rather than multiple-note harmony. I also liked the idea of producing the music through the cassette output port without any specialized hardware to add to the cost. By doing this, I could hook up a small speaker or headphones to the cassette output jack from my TRS-80 and listen to the music production in real time, or I could record the music on the cassette deck for later playback.

Program Operation

The complete jukebox program is shown in the Program Listing. It consists of two dependent parts. The BASIC part draws the jukebox on the display, handles all message output, and accepts the user inputs. The assembly-language portion plays the actual songs, which are coded as string variables in the BASIC portion of the program. The BASIC USR and VARPTR functions communicate between the segments of the program. A song is played by passing the address of the song string variable (provided by the VARPTR

function) to the assembly-language routine via the USR function. The assembly-language music routine then plays the song coded into the string variable until either the string ends or I press the CLEAR key. At that time, control passes back to the BASIC program for additional user interface.

The program is fully commented to make it easy to understand how the BASIC portion of the program operates. The assembly-language music routine, however, is fairly complex, and for that reason, I have not presented it. It would take a full article to completely explain its operation. The following section explains everything you need to know to use the music routine in the jukebox program.

Number	Mnemonic	Function
1	H	Raises current octave by 1
2	I	Reinitializes the music routine to normal initial conditions triplets = off tone quality = normal stacatto = off range = normal octave = normal tempo = fast
3	L	Lowers current octave by 1
4	N	Selects the normal fast tempo
5	Q	Toggles the tone quality from normal to quality 1 and from quality 1 to normal at each occurrence
6	S	Toggles stacatto on/off at each occurrence
7	T	Toggles triplets on/off at each occurrence
8	U	Selects normal range while in the bass range
9	V	Selects bass range
10	W	Selects slower tempo
11	X	Marks alternate ending notes to be played the last time through a repeat
12	<	Marks the start of a repeat phrase. Must be followed immediately by a single digit number from 2–9 specifying the repeat count
13	>	Marks end of a repeated phrase

Each of these functions is activated by placing the mnemonic into the string variable (song) at the position at which the function is required.

Table 1. *Available music functions*

The Assembly-Language Music Routine

The music routine can play any song that can be expressed in whole notes, half notes, eighth notes, and rests. Currently, there are two tempos and two voices available, along with the ability to repeat any phrase up to nine times with an alternate ending available the last time through. This routine converts songs written in pseudo music notation (which are expressed as string variables in the BASIC program) into musical output which is available for listening at the cassette port.

The program has a five-octave range, including sharps and flats, which is broken up into a normal and a bass range (two octaves lower than the normal range). In both of these ranges, there are three octaves available which make the highest bass range equal to the lowest normal range; hence, the five-octave range. Each octave starts with C as the lowest note and B as the highest. Table 1 shows all of the musical functions available from this routine along with a brief explanation of what they do. Each of these functions is activated by placing the mnemonic into the string variable (song) at the position at which the function is required. Most of the functions listed were used in coding the songs that are currently available from the jukebox program.

Notes and rests are coded by placing the duration immediately after the note or rest specification. The note duration can be followed by a dot to increase the duration 1 1/2 times. The following examples illustrate coding of note and rest durations:

A2 = a half note of A
C#4 = a quarter note of C sharp
D- = a whole note of D flat
R2 = a half rest
E2. = a dotted half note of E

Compare "On Top of Old Smokey," shown in Figure 1, to the song string S$(2,2) to get an idea of how the song coding process is performed. Given a little practice, it becomes very easy to code any song that you might want.

Program Execution

The jukebox program runs on any TRS-80 Model I with a minimum 32K of memory. It should also run on the Model III. You can modify the program to run under Level II BASIC or Disk BASIC by changing two lines in the program. The listing, as shown, is for Disk BASIC with the changes for a cassette system in lines 3090 and 3500. The changes in the program handle the differences in the way the USR function is set up and executed by the two BASICs. This is the only difference in the two versions of the jukebox program.

After you make the program resident, by typing it in or loading it from tape or disk, connect the jack of the auxiliary input of your cassette recorder

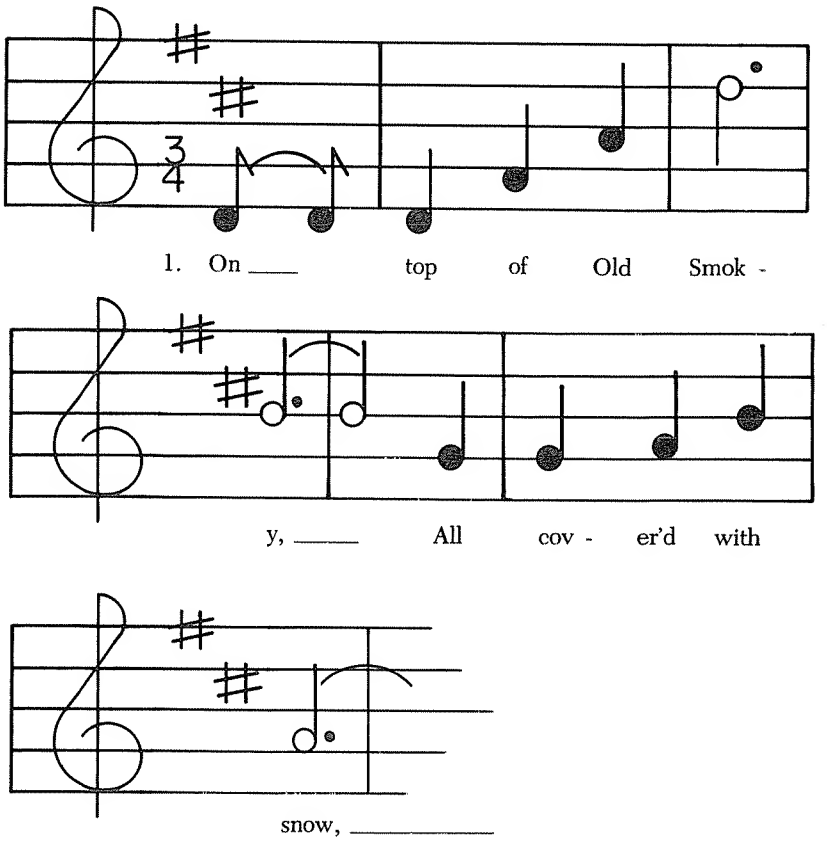


Figure 1. *On Top of Old Smokey*

to a pair of headphones, a small speaker, or an amplifier/speaker combination. When you run the program, the screen clears, and an image of a jukebox appears on the screen along with an initialization message. During this time, the assembly-language routine is being POKEd into memory one byte at a time from the DATA statements in lines 3580–4370. This is a rather slow process, so give it a minute or two. After the complete routine is POKEd into memory, the jukebox buttons begin to blink in a random manner and the program asks you to enter your quarter by hitting any key. At this time, the titles of the first five songs appear in the upper portion of the jukebox with a selection message on the bottom display line. Select a song by pressing a letter (A or B) followed by a number (1–5) corresponding to the song title you want. The INKEY$ function is used throughout so you do not have to press ENTER. If you hit the R key, the song titles scroll upward to display the

other five songs that are available. The R key can be used whenever needed to view the alternate titles. Table 2 shows the 10 song choices. These songs are hardcoded as string variables into the BASIC program in lines 2760–3060.

Selection	String Variable	Song
A 1	S$(1,1)	Yesterday
A 2	S$(1,2)	Scales Demo
A 3	S$(1,3)	Greensleeves Theme
A 4	S$(1,4)	Function Demo
A 5	S$(1,5)	Joy to the World
B 1	S$(2,1)	O Come All Ye Faithful
B 2	S$(2,2)	On Top of Old Smokey
B 3	S$(2,3)	Michelle
B 4	S$(2,4)	Girl from Ipanema
B 5	S$(2,5)	Aquarius

All songs are coded as two-dimensional string variables, S$(X,Y). Songs are played by using the BASIC VARPTR function to point the assembly-language routine at the position in memory occupied by the string variable of the song to be played.

Table 2. *Song selections*

Once you have made a valid selection, the bottom display line shows the title of the song you selected and, after a short delay, the song plays. When the song is over, the jukebox buttons start to blink and the screen asks for a new selection. If you want to end a song prematurely, press the CLEAR key.

To change the songs that the jukebox can play, code the desired song using the procedure given earlier, replace one of the string variables corresponding to an existing song, and replace the title of the song in the DATA statements at the end of the program.

This program, with minor modifications, could form the nucleus of a musical education program for children. You can use it for just about any application that requires musical output. Quite a few dull education and game programs could be enhanced by the addition of fanfares and the like, not just buzzes for wrong or right answers. The uses for this program are almost unlimited.

Program Listing

Encyclopedia Loader™

```
1000 REM  ****************************************
1010 REM  ***                                  ***
1020 REM  ***    TRS-80 JUKE BOX PROGRAM       ***
1030 REM  ***               BY                 ***
1040 REM  ***       CRAIG A. LINDLEY           ***
1050 REM  ***                                  ***
1060 REM  ****************************************
1070 REM
1080 CLS
1090 REM  SET MEMORY SIZE TO PROTECT MUSIC ROUTINE
1100 POKE 16561,223 :
     POKE 16562,188
1110 CLEAR 100
1120 REM  JUMP TO MAIN PROGRAM - OVER THE SUBROUTINES
1130 GOTO 1870
1140 REM
1150 REM  ****************************************
1160 REM           PROGRAM SUBROUTINES
1170 REM  ****************************************
1180 REM
1190 REM  SUBROUTINE #1 - CLEAR OFF BOTTOM DISPLAY LINE
1200 PRINT @960, CHR$(30);
1210 RETURN
1220 REM
1230 REM  SUBROUTINE #2 - DISPLAY SONG TITLES AND TOGGLE FLAG
1240 GOSUB 1340
1250 PRINT @81,"T R S - 8 0      J U K E   B O X";
1260 P% = 213:
     :
     ' FIRST LINE DISPLAY POSITION
1270 FOR I% = 1 TO 5
1280  PRINT @P%,T$(F%,I%);
1290  P% = P% + 64
1300  NEXT
1310 IF F% = 1
      THEN
       F% = 2 :
      ELSE
       F% = 1 :
       ' TOGGLE DISPLAY FLAG
1320 RETURN
1330 REM
1340 REM  SUBROUTINE #3 - CLEAR DISPLAY PORTION OF JUKE BOX
1350 P% = 213
1360 FOR I% = 1 TO 5
1370  PRINT @P%, STRING$(28,32);
1380  P% = P% + 64
1390  NEXT
1400 RETURN
1410 REM
1420 REM  SUBROUTINE #4 - LOAD MUSIC ROUTINE INTO MEMORY
1430 PRINT @960,"INITIALIZING - ONE MINUTE PLEASE";
1440 FOR I% = 1 TO 791
1450  AD% = - 1 * (17185 - I%)
1460  READ D% :
      POKE AD%,D%
1470  NEXT
1480 REM  CLEAR OFF BOTTOM DISPLAY LINE
1490 GOSUB 1190
1500 RETURN
1510 REM
1520 REM  SUBROUTINE #5 - BLINK BUTTONS UNTIL KEY IS PRESSED
1530 IN$ = INKEY$:
     IF IN$ < > ""
      THEN
       RETURN
1540 Y% = RND(2) :
     ' SELECT RANDOM BUTTONS TO BLINK
```

```
1550 X% = RND(5)
1560 P% = A%(Y%,X%)
1570 REM  TURN BUTTON OFF
1580 GOSUB 1700
1590 FOR X% = 1 TO 200 :
      ' DELAY FOR VIEW
1600  NEXT
1610 REM  TURN BUTTON BACK ON
1620 GOSUB 1670
1630 FOR X% = 1 TO 20
1640  NEXT
1650 GOTO 1530
1660 REM
1670 REM  SUBROUTINE #6 - DRAW BUTTON AT SELECTED POINT
1680 PRINT @P%, CHR$(191); CHR$(191); CHR$(191);
1690 RETURN
1700 REM  SUBROUTINE #7 - BLANK OUT BUTTON AT SELECTED POINT
1710 PRINT @P%, CHR$(32); CHR$(32); CHR$(32);
1720 RETURN
1730 REM
1740 REM  SUBROUTINE #8 - DRAW DUAL WIDTH LINES ROUTINE
1750 SET(A,Y):
     SET(A + 1,Y):
     SET(B,Y):
     SET(B + 1,Y)
1760 RETURN
1770 REM
1780 REM  SUBROUTINE #9 - PLACE VERTICAL LABEL ROUTINE
1790 FOR I% = 1 TO LEN(BT$)
1800  CH$ = MID$(BT$,I%,1)
1810  PRINT @P%,CH$;
1820  P% = P% + 64
1830  NEXT
1840 RETURN
1850 REM
1860 REM  ****************************************
1870 REM          START OF THE MAIN PROGRAM
1880 REM  ****************************************
1890 REM
1900 REM  INITIALIZE DISPLAY FLAG
1910 F% = 1
1920 DIM A%(2,5),S$(2,5),T$(2,5)
1930 REM  BUTTON POSITION ARRAY DATA
1940 A%(1,1) = 711:
     A%(2,1) = 839
1950 A%(1,2) = 723:
     A%(2,2) = 851
1960 A%(1,3) = 735:
     A%(2,3) = 863
1970 A%(1,4) = 747:
     A%(2,4) = 875
1980 A%(1,5) = 759:
     A%(2,5) = 887
1990 REM
2000 REM  DRAW TOP OF JUKE BOX
2010 FOR X = 26 TO 101
2020  SET(X,0)
2030  NEXT
2040 REM
2050 REM  DRAW SLOPING SIDES
2060 X = 24
2070 FOR Y = 0 TO 12
2080  A = X - (Y * 2)
2090  B = X + 78 + Y * 2
2100  GOSUB 1740
2110  NEXT
2120 A = 0
2130 FOR Y = 21 TO 27
2140  B = 126 - A
2150  A = A + 2
```

Program continued

```
2160  GOSUB 1740
2170  NEXT
2180 REM
2190 REM  DRAW VERTICAL SIDES
2200 FOR Y = 0 TO 28
2210  A = 24
2220  B = 102
2230  GOSUB 1740
2240  NEXT
2250 FOR Y = 12 TO 43
2260  A = 0
2270  B = 126
2280  GOSUB 1740
2290  NEXT
2300 REM
2310 REM  DRAW HORIZONTAL LINES
2320 FOR X = 2 TO 125
2330  SET(X,27):
      SET(X,28):
      SET(X,43)
2340  NEXT
2350 REM
2360 REM  DRAW BUTTONS
2370 FOR I% = 1 TO 5
2380  P% = A%(1,I%)
2390  GOSUB 1670
2400  P% = A%(2,I%)
2410  GOSUB 1670
2420  NEXT
2430 REM
2440 REM  DRAW MONEY SLOT
2450 FOR Y = 34 TO 38
2460  SET(120,Y):
      SET(121,Y)
2470  NEXT
2480 REM
2490 REM  DRAW LABELS
2500 PRINT @647,1; TAB(19)2; TAB(31)3; TAB(43)4; TAB(55)5; TAB(59)".2
     5";
2510 PRINT @707,"A";
2520 PRINT @835,"B";
2530 PRINT @892,"$";
2540 REM
2550 REM  PRINT TITLES
2560 PRINT @219,"T R S - 8 0";
2570 PRINT @284,"JUKE BOX";
2580 PRINT @415,"BY";
2590 PRINT @472,"CRAIG A. LINDLEY";
2600 REM
2610 REM  PLACE VERTICAL LABELS
2620 BT$ = "TOP 40"
2630 P% = 199
2640 GOSUB 1780
2650 BT$ = "HITS"
2660 P% = 312
2670 GOSUB 1780
2680 REM
2690 REM  POKE MUSIC ROUTINE INTO MEMORY
2700 GOSUB 1420
2710 REM
2720 REM  ****************************************
2730 REM     THESE STRINGS ARE THE CURRENT SONGS
2740 REM  ****************************************
2750 REM
2760 REM  THIS SONG IS YESTERDAY
2770 S$(1,1) = "W<2L<2G8F8F2.R8A8B8HC#8D8E8F8E8.D2.R8D8D8C8LB-8A8G8B-
     4A8A4.G4F4A8G2D8F4A8A2.>SQA2A2HD4E4F4E8D8E4.D8C4D8LAA2A2HD4E4F4E
     8D8E4.D8C4E4FSQ >"
2780 REM
2790 REM  THIS IS A DEMO OF SCALES
```

```
2800 S$(1,2) = "<2V<2LCDEFGABHCRCDEFGABHCRCDEFGABB#I>Q>"
2810 REM
2820 REM  THIS SONG IS GREENSLEEVES
2830 S$(1,3) = "W<2E4G2A4B4.HC8LB4A2F#4D4.E8F#4G2E4E4D#4E4F#2D#4LB2HE
     4G2A4B4.HC8LB4A2F#4D4.E8F#4G4F#4E4D#4.C#8D#4E2E4E2.QSHD2.D4.C#8L
     B4A2F#4D4.E8F#4G2E4E4.D#8E4F#2D#4LB2.HHD2.D4.C#8LB4A2F#4D4.E8F#4
     G4F#4E4D#4.C#8D#4E2.E2.QS>"
2840 REM
2850 REM  THIS SONG IS A DEMO OF THE AVAILABLE FUNCTIONS
2860 S$(1,4) = "<2H<3TCDEF<8G8A8>GRQCDEF<8G8A8>GQRTL>HV>"
2870 REM
2880 REM  THIS SONG IS JOY TO THE WORLD
2890 S$(1,5) = "L<2SHD2C#4.LB8A2.G4F#2E2D2.A4B2.B4HC#2.C#4DD2.D4D4C#4
     LB4A4A4.G8F#4HD4D4C#4LB4A4A4.G8F#4F#4F#4F#4F#4F#4F#8G8A2.G8F#8E4E4E
     4E8F#8G2.F#8E8D4HD2LB4A4.G8F#4G4F#2E2D2D2>"
2900 REM
2910 REM  THIS SONG IS O COME ALL YE FAITHFUL
2920 S$(2,1) = "WV<2A4A2E4A4B2E2HC#4LB4HC#4D4C#2LB4A4A2G#4F#4G#4A4B4H
     C#4LG#2F#4.E8E2E2HE2D4C#4D2C#4LB4HC#4LA4B4G#4.F#8E4A4A4G#4A4B4A2
     E4HC#4C#4LB4HC#4D4C#2LB4HC#4D4C#4LB4A4G#2A4HD4C#2.LB4.A8A2.>"
2930 REM
2940 REM  THIS SONG IS ON TOP OF OLD SMOKY
2950 S$(2,2) = "VWS<3D8D8D4F#4A4HD2.LB2.B2G4G4A4B4A2.A2.A2D8D8D4F#4A4
     A2.E2.E2F#8F#8G8G8F#4E4D2.D2.D4R8>"
2960 REM
2970 REM  THIS SONG IS MICHELLE
2980 S$(2,3) = "VWS<2<2A2A2R4B-4F2E4A4E4E-4D4F4A-4F4E2D4F4E>QA4THD4C4
     LA4HD4C4LA4THE4D2.R8LA8B-8A8B-4F4FR8A8A8A8HD4LA4G8F8E4.E8G4A4A4A4
     4A4A4A4A4A4A2G4F4EQ>"
2990 REM
3000 REM  THIS SONG IS THE GIRL FROM IPANEMA
3010 S$(2,4) = "VW<2S<2G4.E8E4D8G8G4E8E8E8E8D8G8G4E4E4D8G8G8G8E8E8E8E
     8D8F8F8D4D8D8D8C8E8E8C4C8C8C8LB-4HR8C2.C2R2>SFTF4G-4F4E-4F4E-4TD
     -4.E-8E-2E-2.R8G8GTG4A-4G4F4G4F4TE-4.E8F2F2.R8A8A2.TAB-4A4G4A4G4
     TF4.G8G2G2TR4A4B-4HC4LC4D4E4F4G4TG#2.A4TB-4LB-4HC4D4E4F4TF#2.R4>
"
3020 REM
3030 REM  THIS SONG IS AQUARIUS
3040 S$(2,5) = "TVW<2A4B4HCC4D4C8LB8A8G8AA2.G4A4B4B2B4A4A4G4AA2.B4HCC
     4D4C8LB8A8G8G4A2.A2R4G4AR4B4B4B4HC4C4QD4C4E4D4C4LB-4B-2B-4A4B-4H
     C4D4C4LB-2B-4A4B-4HC4D2DD2.C4D4F4G2GG2.F4G4F4D2DDLQ>"
3050 REM
3060 REM  ****THIS IS THE END OF THE SONGS ****
3070 REM
3080 REM  DEFINE USRO TO BE THE MUSIC ROUTINE
3090 REM  FOR NON DISK POKE 16526,224 : POKE 16527,188
3100 DEF USRO = &HBCE0
3110 REM
3120 REM  ASSIGN SONG TITLES TO ARRAY
3130 FOR I% = 1 TO 5
3140  READ D$
3150  T$(1,I%) = D$
3160  NEXT
3170 FOR I% = 1 TO 5
3180  READ D$
3190  T$(2,I%) = D$
3200  NEXT
3210 REM
3220 REM  SONG PLAYING LOOP
3230 REM
3240 REM  CLEAR OFF BOTTOM DISPLAY LINE
3250 GOSUB 1190
3260 PRINT @960,"PLEASE ENTER YOUR QUARTER";
3270 GOSUB 1520
3280 REM  DISPLAY SONG TITLES
3290 GOSUB 1230
3300 REM  CLEAR OFF BOTTOM DISPLAY LINE
3310 GOSUB 1190
3320 PRINT @960,"SELECT SONG (R - ROLLS SONG TITLES) - EX. A3:";
3330 REM  BLINK BUTTONS UNTIL SELECTION IS MADE
3340 GOSUB 1520
```

Program continued

```
3350 IF IN$ = "R"
      THEN
       GOTO 3280
3360 REM  FIRST ENTRY MUST BE A OR B
3370 IF IN$ < > "A" AND IN$ < > "B"
      THEN
       GOTO 3300
3380 PRINT @1007,IN$;
3390 IF IN$ = "A"
      THEN
       R% = 1 :
      ELSE
       R% = 2
3400 IN$ = INKEY$:
     IF IN$ = ""
      THEN
       GOTO 3400
3410 PRINT @1009,IN$;
3420 C% = VAL(IN$)
3430 IF C% < 1 OR C% > 5
      THEN
       GOTO 3300
3440 REM  CLEAR OFF BOTTOM DISPLAY LINE
3450 GOSUB 1190
3460 PRINT @960,"NOW PLAYING SELECTION:  "; MID$(T$(R%,C%),5);
3470 FOR I% = 1 TO 1000
3480  NEXT
3490 REM  NOW PLAY THE SELECTED SONG
3500 REM  FOR NON DISK A=USR(VARPTR(S$(R%,C%)))
3510 A = USR0( VARPTR(S$(R%,C%)))
3520 GOTO 3300
3530 REM
3540 REM  ****************************************
3550 REM              PROGRAM DATA
3560 REM  ****************************************
3570 REM
3580 DATA 195,191,190, 12, 64, 16,106,188, 67, 0
3590 DATA 68, 2, 69, 4, 70, 5, 71, 7, 65, 9
3600 DATA 66, 11, 82, 15, 32, 34, 36, 38, 40, 43
3610 DATA 45, 48, 51, 54, 57, 60, 64, 68, 72, 76
3620 DATA 81, 85, 90, 96,101,107,114,121,128,135
3630 DATA 143,152,161,171,181,192,203,215,228,241
3640 DATA 255, 21, 23, 24, 25, 27, 29, 30, 32, 34
3650 DATA 36, 38, 40, 43, 45, 48, 51, 54, 57, 60
3660 DATA 64, 67, 71, 76, 81, 85, 90, 95,101,107
3670 DATA 114,121,128,135,143,152,161,170,243,230
3680 DATA 217,205,193,182,172,162,153,145,137,129
3690 DATA 121,115,108,102, 96, 91, 86, 81, 76, 72
3700 DATA 68, 64, 60, 57, 54, 51, 48, 45, 43, 40
3710 DATA 38, 36, 34, 32, 30,245,231,218,206,195
3720 DATA 184,173,164,155,145,137,130,123,115,110
3730 DATA 103, 98, 92, 87, 82, 78, 73, 69, 65, 62
3740 DATA 58, 55, 52, 49, 46, 43, 41, 39, 37, 35
3750 DATA 33, 31,233,206,180,156,133,111, 90, 71
3760 DATA 53, 36, 34, 32, 30, 28, 27, 25, 24, 22
3770 DATA 21, 20, 19, 18, 17, 16, 15, 14, 13, 12
3780 DATA 12, 11, 10, 10, 9, 9, 8, 8, 7,255
3790 DATA 255,255,255,255,255,255,255,255,254,240
3800 DATA 227,214,202,191,180,170,161,152,142,135
3810 DATA 127,120,113,107,101, 96, 91, 85, 80, 76
3820 DATA 71, 68, 64, 61, 57, 54, 13, 72, 73, 76
3830 DATA 78, 81, 83, 84, 85, 86, 87, 88, 60, 62
3840 DATA 18,190,149,190, 4,190,157,190,254,189
3850 DATA 185,190,163,190,136,190,115,190, 82,190
3860 DATA 71,190, 47,190, 36,190, 33,229,188,203
3870 DATA 230,201,225,241,245, 61, 32, 5, 33,229
3880 DATA 188,203,214,195,229,190,225,241, 61, 40
3890 DATA 4,209,213,245,233, 33,229,188,203,150
3900 DATA 241,195,229,190, 33,227,188,126,254, 24
3910 DATA 200,198, 12,119,201,221, 33, 66,189,253
```

```
3920 DATA 33,248,188, 33,228,188, 54, 64, 33,227
3930 DATA 188, 54, 12, 33,229,188, 54, 0,201, 33
3940 DATA 227,188,126,254, 0,200,214, 12,119,201
3950 DATA 33,229,188,203,134,201,229, 78, 6, 0
3960 DATA 35,237,177,225,121, 40, 2,175,201, 78
3970 DATA 6, 0, 35, 9, 7, 79, 9,126, 35,102
3980 DATA 111,241,233, 33,229,188,203,126, 32, 7
3990 DATA 221, 33,140,189,203,254,201,221, 33, 66
4000 DATA 189,203,190,201, 33,229,188,203, 78, 32
4010 DATA 3,203,206,201,203,142,201,225, 26, 19
4020 DATA 213,230, 15,245,233, 33,229,188,203,198
4030 DATA 201, 33,228,188, 62, 64,190, 32, 7,253
4040 DATA 33, 29,189, 54, 43,201,253, 33,248,188
4050 DATA 54, 64,201, 33,229,188,203,166,201,205
4060 DATA 127, 10,237,115,230,188, 35, 94, 35, 86
4070 DATA 62, 4,211,255,221, 33, 66,189,253, 33
4080 DATA 248,188, 33,228,188, 54, 64, 33,227,188
4090 DATA 54, 12, 33,229,188, 54, 0,205, 91, 3
4100 DATA 33, 1, 0,254, 31, 40, 7, 26,183, 32
4110 DATA 10, 33, 0, 0,237,123,230,188,195,154
4120 DATA 10, 19, 33,229,190,229, 33,232,188, 1
4130 DATA 16, 0,237,177, 40, 7, 33,214,189,205
4140 DATA 88,190,201, 78, 26,254, 35, 32, 3, 12
4150 DATA 24, 5,254, 45, 32, 3, 13, 19, 26, 6
4160 DATA 16,254, 50, 32, 4, 6, 8, 24, 14,254
4170 DATA 52, 32, 4, 6, 4, 24, 6,254, 56, 32
4180 DATA 4, 6, 2, 19, 26,254, 46, 32, 6,120
4190 DATA 203, 47,128, 71, 19, 33,229,188,203, 86
4200 DATA 192, 62, 15,185,202,211,191, 33,227,188
4210 DATA 126,129, 33,160,191,119, 33,134,191,119
4220 DATA 198, 37, 33,186,191,119, 33,229,188,203
4230 DATA 70, 32, 2,203, 56,197,205,132,191,193
4240 DATA 16,249, 33,229,188,203, 78,200, 6, 8
4250 DATA 205,240,191, 16,251,201,253, 70, 0, 58
4260 DATA 229,188,203,103, 40, 4,203, 56,203, 56
4270 DATA 197,205,154,191,193, 16,249,201, 62, 5
4280 DATA 211,255,221, 70, 0, 58,229,188,203,103
4290 DATA 40, 10, 72, 16,254, 65, 16,254, 65, 16
4300 DATA 254, 65, 16,254, 62, 4,211,255,221, 70
4310 DATA 0, 58,229,188,203,103, 40, 10, 72, 16
4320 DATA 254, 65, 16,254, 65, 16,254, 65,120,214
4330 DATA 18, 71, 16,254,201,205,219,191, 16,251
4340 DATA 195,118,191,197, 33,228,188, 70, 58,229
4350 DATA 188,203, 71, 32, 2,203, 56,205,240,191
4360 DATA 16,251,193,201,197, 6,244, 16,254,193
4370 DATA 201
4380 REM
4390 REM  SONG TITLE DATA
4400 DATA "A-1 YESTERDAY","A-2 SCALES","A-3 GREENSLEEVES","A-4 DEMO",
     "A-5 JOY TO THE WORLD"
4410 DATA "B-1 O COME ALL YE FAITHFUL","B-2 ON TOP OF OLD SMOKY","B-3
      MICHELLE","B-4 GIRL FROM IPANEMA","B-5 AQUARIUS"
```

GRAPHICS

Instant Graphics for Everyone
Screen Editor for Graphics Creations

Instant Graphics for Everyone

by Ralph Vickers

Instant Graphics for Everyone is a utility which helps you write machine-code graphics in BASIC. It completely avoids the complications of loading balky SYSTEM tapes and wrestling with the intricacies of EDTASM and its weird numbering system. You can write in BASIC and still create instant video screen graphics. If you know the difference between PEEK and POKE, you know how to draw instant graphics on your screen in just about any manner you want.

To begin, you need a copy of Instant Graphics for Everyone. (See Program Listing.) Enter 30000 as the memory size. All references in the article pertain to a 16K system. If you have a different memory size, you must scale accordingly.

Via the menu, find your way to the Program Test routine. Once you have chosen this routine, ignore the first display and follow the instructions of the Caution display. The result is a horizontal graphics bar, stretching across the screen, that appears in an instant. You could have done this just about as fast with PRINT STRING$(60,140). But how would you like to bisect this display with a vertical line from the top to the bottom of the screen?

First, you must understand two facts about your computer.

1) It has 32768 locations for storing information, numbered from 0 to 32767. These numbers are called memory addresses.

2) There are a few temporary storage places called registers. When you pull a bit of information out of a memory address, it goes into one of the registers. These registers are called A,B,C,D,E, and there's a double-decker register named HL.

When you look at machine-code programs you see annotations like 3E, which is the hexadecimal representation of decimal 62. We'll use 62 for our machine-code composition. To the computer, they both mean the binary number 00111110.

Return to the program index and choose the Write Data routine. As the display says, you are now in the Command Mode and you are looking at DATA line 10000 which contains 13 ordinary numbers. That's the machine code you used to draw the horizontal bar. You are now going to write in the vertical line. Start your first entry as:

```
10010 DATA 62,
```

There are about 700 numbers that your computer understands as specific commands when you talk to it in machine language. Number 62 tells the computer: Load A register with.

Here you have an opportunity to exercise your creative impulses. From a display of the TRS-80 graphics characters, pick a CHR$(nnn) graphics number that strikes your fancy. I'll use 191. Enter this or anything between 129 and 190 as the second number in your DATA line. At this point, you have complete flexibility to draw any kind of line you want. If you're a little confused, enter in Command Mode PRINT CHR$(191) and you'll see what I'm talking about.

You now have a DATA line with two numbers, 62 and 191. When the computer gets a chance to read the numbers, it understands them as two commands saying: Load A register with the equivalent of CHR$(191).

It is necessary to give the computer a position at which to print, and this is a little more complicated to say in machine language than in BASIC. To select the top center of the screen to start the display, in BASIC you would say: PRINT @ 32. In machine language, you have to express the position as a memory address. The number 33 loads this into the HL register.

There are 1024 memory addresses reserved for storing information to be displayed to your video screen. They are numbered from 15360 to 16383. To find the address corresponding to position 32 on your video screen, add 32 to 15360. 15392 is the position number the computer needs, but it is a decimal number. You must give it to the computer in machine code.

There's an easy solution at hand. Run the program and go to the Screen Location routine. Enter 32 as the decimal value of the memory location. The readout says: Decimal Memory Location 15392, but it adds: Express as 32 60.

If you do the following sum, (60 × 256) + 32, the answer you get is 15392.

Go back to the Write Data routine and add 32 and 60 to your DATA line. Now you have:

```
10010 DATA 62,191,33,32,60
```

This DATA line is saying in machine language: PRINT @ 32, CHR$(191). Remember that the computer processes and executes 62,191,33,32,60 much faster than it can carry out a CHR$ statement.

You see by now that, in regard to 62 and 33, you don't have any option in this system of machine language. You do, however, have some choice in regard to the other three numbers. You can substitute any graphics character from 129 to 190 for the 191 I chose. If you play around with the Screen Location routine, you will discover 1023 other combinations of numbers for which you can substitute those last two numbers in your DATA line.

You now have most of the basic tools needed to place any graphics character anywhere on the screen using machine language.

Machine-language programmers place REM remarks at the end of each program line. You, too, can have this accessory. Instead of the format you have used, you could set up your program to look like this:

```
10010 DATA 62 : 'Load A register
10020 DATA 191 : 'CHR$(191) @ 32
```

You now have a graphics character almost at the top of the screen, but the deal was to run a line all the way down. The next magic number you need is 6, which is a machine-code instruction to tell the computer: Load the B register with a number. The number is the number of times you want to repeat the 191 graphics character. There are 16 positions down the screen. There will be a prompt at the bottom of the screen, so let's settle for 14. We now add 6 and 14 to the DATA line. Think of these two numbers as a piece of a BASIC FOR-NEXT statement. What you have done is drop a note in the B register saying: Perform 14 loops.

In reality, however, you are going to build with one more command yet to be revealed, a set of instructions exactly equivalent to the statement FOR A = 14 TO 1 STEP − 1. You will do this FOR-NEXT statement the hard way because the computer has an automatic test for zero but doesn't have one for 14.

If you were writing this program in BASIC, you would have to make some provision for incrementing the @32 position by 64 each time you went through one of the 14 loops. The same logic applies in machine language. The number which initiates this procedure is 17. Number 17 tells the computer: Load the DE register with a two-part number. The code you want is:

```
17,64,0
```

That isn't as complicated as it looks. You know what the 17 means. If the 64,0 isn't immediately clear, refresh your memory by reviewing how you wrote 15392 into the program. The sum is (0 × 256) + 64 = 64.

To write graphics, you never need a number here larger than 255. The number 256 is expressed as 0,1.

The program has now grown to:

```
62,191,33,32,60,6,14,17,64,0
```

You have arrived at the beginning of the one tricky part. Pay close attention because if you get this part wrong your program will crash. When a machine-code program crashes, you usually find yourself staring at MEMORY SIZE?

The next instruction to put down is 119, which means: PRINT the contents of A register at the screen position stored in the HL register. Remember that the HL register can hold two pieces of information. Also keep in mind that you loaded the A register with CHR$(191) and the HL register with screen position 15392, expressed as 32,60. If this part of the program was running, your video screen would light up at position 32 with the graphics character you chose. At memory location 119, you begin a FOR-NEXT type

of loop. In other words, to PRINT CHR$(191) on the screen 14 times, repeat the 119 instruction 14 times.

To write this routine in BASIC you would put in an instruction to increment the screen position by 64 to prepare for the next PRINT instruction. You accomplish this feat by adding 25 to the DATA line. This number tells the computer: Increase the screen position value in the HL register (which is now 15392) by the value in the DE register (which is 64).

To whet your appetite for further exploration, I'll mention that the number 9 would put the contents of a BC register (if you had one set up) into the HL register.

The next detail you must attend to is to reduce the counter by 1. Number 5 takes care of that.

Finally, you must set up the part of the loop that returns to the 119 command. You also need to keep an eye on the decrementing loop value so that when it is reduced to zero, the computer moves from the loop to the next program instruction. 194 takes care of that chore. Your DATA line now looks like this:

```
10010 DATA 62,191,33,32,60,6,14,17,64,0,119,25,5,194
```

Now comes the tricky part I warned you about. 194 tells the computer to loop back, but the computer cannot figure out where to go. You want it to return to memory address 119 and execute the instructions it finds there. To direct the computer to 119, you must specify the memory address number. Table 1 lists the machine codes and their functions.

If you were writing conventional machine-language code, when you wrote the 119 line you would add LOOP1. On the line containing 194, you would specify LOOP1 again and the computer would find its own way back. This is really no problem. It just means an extra step.

Add two zeros to the end of the DATA line. This is to provide spaces for two numbers you must now insert. Run the program and go to the Program Test. You should now have on your screen a display that starts off with: BS = 30. BS means bytes. In other words, BS indicates the number of DATA numbers you have in your program, assuming you didn't kill off the 10000 DATA line I originally provided. If BS doesn't equal 30, check your DATA line for an entry error.

ML equals the starting memory location of your machine-code program. It has already been POKEd into high memory. You can confirm this by checking Table 1. The first item should be 32737 62 if you are using a 16K system. This means that the number 62 is POKEd into memory address 32737.

Check the third row, third column on the displayed table. There should be a 194 there. It's flagged with an asterisk to warn you that you must triple-check the following two numbers which are now wrong. Go back to the preceding 119. You should find it at 32744. Write that address down.

The last item on your screen should be a 119 at 32760. Jot that memory address down too. That's the location you want the next 194 instruction to send the computer back to. Pull up the remainder of the memory location table. There's your 194 flagged, plus the two zeros behind it.

The 201 that has mysteriously appeared says: Return to BASIC. This 201 is tucked out of your way at program line 12990. There is also a −1 there. It has nothing to do with the machine code. It is just a READ flag.

You now have written down two memory addresses that you must translate into two-part numbers that the computer can understand. Find your way to the Memory Location routine. Get a readout on both 32744 and 32760 and return to your DATA statement.

Code		Function
62		Load A register with
149	(Variable)	CHR$(149)
33		Load HL register with
32	(Variable)	screen location (first part)
60	(Variable)	screen location (second part)
6		Load B register with increment of
14	(Variable)	number of loops
17		Load increment into DE register of
64	(Variable)	increment number (first part)
0		increment number (second part)
119		PRINT contents of A register at screen position in HL register
25		Add DE value to screen location in HL
5		Reduce value in B register by one (loops)
194		Loop back to memory location of 119
228	(Variable)	memory location number (first part)
123	(Variable)	memory location number (second part)
201		Return to BASIC program

Table 1. *Machine codes and functions*

At line 10000, immediately after 194, delete the 248 and 127 and substitute the new numbers you got, 232 and 127, in that order. The two 32760 numbers go into the slots you reserved with the two zeros at the end of your DATA line.

It is essential that you thoroughly understand what you've just done. The two-part numbers immediately behind the 194s are the memory addresses corresponding to the 194's respective 119s. It is even more important to remember that, if you add even one more DATA number to this machine-language routine, you create a whole new ball game in regards to the relationship between the 194s and the 119s. In this case, all the return addresses have to be recalculated because, when you add a new number, the whole

program is shunted down one memory address. If you were working up from the bottom of memory, it would be no problem, but you are working down from the top of memory.

In case you have made an error, make a tape of this program right now. Now go to the Program Test menu option. Plunge recklessly through the caution barrier. If all goes well, your video screen will light up instantaneously with a big cross. When you have admired this display long enough, proceed to the next display. Your next task is to fit this machine-language routine into a BASIC program.

The Set Memory At number displayed is the answer to give your computer's MEMORY SIZE? query when you use this routine in one of your programs. The number should be 32736, one less than the starting address of this routine. You can answer MEMORY SIZE? with a number lower than the one given, but not a higher number.

If you insert line 100 exactly as it appears, this tells the computer the memory address at which your graphics routine starts.

The POKE statement numbers 16526 and 16527, which are also memory addresses, never change. The other two numbers, which should be 255 and 127, vary according to the length of your machine-code routine. To make this clearer, do the following sum:

```
(127 * 256) + 225 = 32737
```

Consider the UR = USR(0) statement a specialized GOSUB that branches your program to the memory location specified by the POKE 16526 and 16527 statements. If you have two or more subroutines you want to access in the same program, you can use the same or another USR(0) statement, but immediately before it you must change the address to go to with new POKE 16526 and 16527 statements.

Finally, on the next display, you are given the FOR-NEXT numbers you need to POKE your DATA into the right memory slots. To review the powerful tool you now have, refer to Table 1.

You have learned eight versatile machine-language codes (just another 692 to go) which you can use to perform a variety of graphics wonders. Experiment by shifting the locations of the lines around the screen, making them shorter and longer.

Bit by bit, you will pick up other number codes you can use. For instance, browse through machine-language programs. You will start spotting useful lines that look like this:

```
7EBC 23   00310  INC   HL   ;INCREMENT HL REGISTER BY 1
```

This reveals a special code which increments the screen position in your HL register by 1. Let's analyze this line of machine code so you can learn how to extract useful information from this type of code.

7EBC is the memory address at which the first piece of information con-

tained in this line is stored. Some lines of machine code have more than one piece of information in them, but each one is counted. The next listed memory address has taken everything into account. Otherwise, these numbers run consecutively.

To decipher this code, go to the Instant Graphics for Everyone, Hex Translation routine. Key in 7E. Press C to continue the operation and key in BC. The result is memory address 32444.

The next part is the interesting bit. See the 23? Translate it into decimal. The answer is 35. That's the number to use whenever you want to increment your current screen position address, in your HL register, by 1.

The 00310 is the assembly-language program line number—the same as in BASIC. INC is the tipoff to what the line does. Some of these mnemonics you can guess: INC for increment, DEC for decrement, ADD for add. But you need a source book to figure out beauties like DJNZ. HL you know, and this is followed by the programmer's REM statement that you should read carefully for clues to line functions. There are 16 different DEC codes. The one written as 2B is 43 decimal. Put 43 into one of your machine-code subroutines, and your HL register screen position drops one notch.

While you're browsing through machine code (actually, it's called assembly-language source code) in magazines or wherever, if you see a column of 23-type annotations between the memory addresses (first column) and the line numbers, you should easily be able to copy the program into decimal DATA statements using the Instant Graphics for Everyone program.

A few hints: Note that those digits are always listed in pairs. An example is CDD1E9, which I borrowed from an actual program listing. Read that as CD D1 E9. Your Hex Translator can tell you that this means 205 209 233.

Program Listing. *Instant Graphics for Everyone*

Encyclopedia Loader

```
 10 CLS
100 CLEAR 60
140 POKE 16553,255
150 ON ERROR GOTO 400
160 A = 128:
    A1 = 16254
190 GOTO 900
200 INPUT "TO PROCEED, KEY > ENTER < ";AN:
    RETURN
210 PRINT "ENTER INSTRUCTIONS ('T' TO TERMINATE) >>>>>"
215 AN$ = "":
    AN = 0
220 AN$ = INKEY$:
    IF AN$ = ""
     THEN
      220:
     ELSE
      AN = VAL(AN$):
      IF AN$ = "T"
       THEN
        800:
       ELSE
        RETURN
230 PRINT @A, CHR$(31);:
    RETURN
259 :
    ' SUBROUTINE TO CONTROL LIST SCROLL
260 AC = PEEK(16417) * 256 + PEEK(16416):
    IF AC > A1
     THEN
      GOSUB 200:
      GOSUB 230
270 RETURN
400 PRINT "SORRY, THERE HAS BEEN A #"; ERR / 2 + 1;" ERROR ON LINE";
     ERL :
    GOSUB 200:
    RESUME 800
800 :
    ' THIS IS AN ABORT RETURN LINE TO RESET VARIABLES
900 CLS :
    PRINT TAB(5)"INSTANT GRAPHICS FOR EVERYONE":
    PRINT STRING$(50,131)
910 PRINT "INDEX"
920 PRINT "HEX TRANSLATION.............#1"
930 PRINT "MEMORY LOCATION.............#2"
940 PRINT "SCREEN LOCATION.............#3"
950 PRINT "PROGRAM TEST................#4"
960 PRINT "WRITE DATA..................#5"
970 PRINT "PRINT#-1, DATA..............#6"
990 PRINT :
    GOSUB 210:
    ON AN GOTO 1090,1400,1500,2000,9990,5000
1090 GOSUB 230
1100 PRINT :
    PRINT "ENTER DIGIT #1 >>>>> ";:
    GOSUB 215:
    PRINT AN$;:
    E$ = AN$:
    PRINT TAB(51)"H"; TAB(57)"D"
1110 D = ASC(AN$):
    IF D < 58
     THEN
      M = (D - 48) * 16:
      GOTO 1130
1120 M = (D - 55) * 16
1130 PRINT "ENTER DIGIT #2 >>>>> ";:
    GOSUB 215:
```

```
     PRINT AN$;
1140 D = ASC(AN$):
     IF D < 58
      THEN
       N = D - 48:
       GOTO 1160
1150 N = D - 55
1160 P = M + N:
     PRINT TAB(50)E$ + AN$;" = ";P:
     IF R = 1
      THEN
       1300
1200 PRINT "('C'=CONTINUE <> 'ENTER'=RESTART <> 'M'=MEM)":
     GOSUB 210
1205 IF AN$ = "M"
      THEN
       1400
1210 IF AN$ = "C"
      THEN
       R = 1
1220 IF R = 1
      THEN
       Q = P:
       GOTO 1100
1230 GOTO 1090
1300 S = (Q * 256) + P:
     R = 0
1310 PRINT TAB(32)Q;" * 256 +";P;" = ";S:
     PRINT
1320 GOTO 1200
1400 BS = 0:
     GOSUB 230:
     PRINT :
     INPUT "ENTER DECIMAL VALUE OF MEMORY LOCATION";ML:
     GOSUB 1410:
     GOTO 1200
1410 MS = INT(ML / 256):
     LS = ML - (MS * 256):
     IF BS > 0 RETURN
1420 PRINT :
     PRINT TAB(20)"EXPRESS AS:",LS,MS:
     PRINT :
     RETURN
1500 GOSUB 230:
     PRINT "TO UTILIZE THIS ROUTINE YOU MUST":
     PRINT "REFER TO THE TRS-80 VIDEO DISPLAY WORKSHEET":
     GOSUB 200
1510 GOSUB 230:
     INPUT "ENTER @ POSITION REQUIRED";AP:
     PRINT
1520 PRINT "DECIMAL MEMORY LOCATION IS"; TAB(32)AP + 15360:
     ML = AP + 15360
1530 GOSUB 1410:
     GOSUB 210:
     GOTO 1510
2000 GOSUB 230:
     PRINT "PROGRAM TEST":
     BS = 0
2010 READ Y:
     IF Y = - 1
      THEN
       2030
2020 BS = BS + 1:
     GOTO 2010
2030 RESTORE
2040 FOR X = 32767 - BS TO 32766
2050  READ Y:
      POKE X,Y:
      NEXT :
     RESTORE
```

Program continued

```
2060 ML = 32767 - BS:
     GOSUB 1410
2070 POKE 16526,LS:
     POKE 16527,MS
2080 GOSUB 3000:
     CLS
2085 UR = USR(0)
2100 PRINT @896,"";:
     GOSUB 200
2110 GOSUB 230:
     PRINT "SET MEMORY AT";32766 - BS;"     NUMBER OF BYTES =";BS
2120 PRINT :
     PRINT "BASIC PROGRAM REQUIRED TO RUN THIS GRAPHIC ROUTINE:":
     PRINT
2130 PRINT "100 POKE 16526,";LS;" :POKE 16527,";MS;:
     PRINT "  :'(NEAR PROGRAM START)"
2140 PRINT "9000 UR=USR(0)   :'(THIS LINE BRANCHES TO GRAPHICS SUBROU
     TINE)"
2150 PRINT "9010   (MACHINE CODE SUBROUTINE RETURNS TO BASIC HERE)":
     PRINT :
     GOSUB 200
2160 GOSUB 230:
     PRINT "ROUTINE TO LOAD MACHINE CODE DATA:":
     PRINT
2170 PRINT "110 FOR X =";ML;" TO";ML + BS
2180 PRINT "120 READ Y: POKE X,Y: NEXT"
2190 PRINT :
     PRINT "NOTE: THE DATA CAN ALSO BE LOADED FROM CASSETTE TAPE":
     PRINT :
     PRINT :
     GOSUB 200:
     GOTO 900
3000 GOSUB 230
3010 PRINT "BS =";BS
3020 PRINT "LS =";LS,:
     PRINT "(MS * 256) + LS =";ML
3030 PRINT "MS =";MS
3035 PRINT "ML =";32767 - BS
3040 L = PEEK(16526):
     PRINT "16526 (LS) =";L
3050 M = PEEK(16527):
     PRINT "16527 (MS) =";M
3070 FOR X = 32767 - BS TO 32766:
      GOSUB 260
3080  M1 = PEEK(X):
      PRINT X;M1;:
      IF M1 = 194
       THEN
        PRINT "*",:
       ELSE
        PRINT "",
3090  NEXT :
     PRINT
3100 GOSUB 200:
     GOSUB 230:
     PRINT ">>>>>>>>>> CAUTION <<<<<<<<<<":
     PRINT "TO PROCEED BEYOND THIS POINT BEFORE YOU ARE READY":
     PRINT "COULD RESULT IN PROGRAM CRASH.":
     PRINT :
     PRINT "TO PROCEED, ENTER 'P'":
     PRINT "ANY OTHER KEY WILL RETURN YOU TO INDEX":
     PRINT :
     GOSUB 210
3105 IF AN$ = "P" RETURN
3110 GOTO 900
5000 GOSUB 230:
     PRINT ">>>>> IS CASSETTE READY TO RECORD? <<<<<":
     PRINT :
     GOSUB 200:
     GOSUB 230
```

```
 5005 IF BS = 0 PRINT "SORRY, YOU MUST GO THROUGH 'PROGRAM TEST' ROUTI
      NE":
      PRINT "TO COUNT BYTES OF DATA":
      PRINT :
      GOSUB 200:
      GOTO 2000
 5010 PRINT "PRINTING DATA"
 5020 FOR X = 1 TO BS:
       READ Y
 5025  PRINT # - 1,Y
 5030  NEXT :
      RESTORE :
      PRINT :
      PRINT "DATA PRINTED":
      PRINT :
      PRINT :
      GOSUB 200:
      GOTO 900
 9990 GOSUB 230:
      PRINT "YOU ARE IN COMMAND MODE":
      PRINT "WRITE DATA ON NEXT AVAILABLE LINE....THEN 'RUN'":
      LIST 10000 - 12980
10000 DATA 62,131,6,61,33,128,61,119,35,5,194,248,127
12990 DATA 201,-1
24999 END
```

Screen Editor for Graphics Creations

by Bruce Douglass

Many programs have been published which enable you to draw and even save your drawings on disk or tape. I have not seen one, however, that is useful for the production of graphics displays, such as logos, that add polish to a professional looking program. I developed SCRNEDIT (see Program Listing) because I needed to develop quick, simple, and attractive graphics displays for my programs.

I felt that a screen editor should include the following:

- Full screen movement of the cursor
- The ability to draw or erase at will
- The ability to link screens easily with machine-language or BASIC programs
- The ability to erase to the end of the page
- The ability to erase and restart
- The use of ASCII characters as well as graphics blocks
- The ability to do inverse graphics easily

SCRNEDIT has these attributes. The arrow keys control the flashing cursor. The graphics are in bit resolution using SET and RESET, complete with a repeating key. The only keys that repeat are the arrow keys. To draw, simply hold down an arrow key. The function repeats if you hold down the key. If you do not like a point or a line, the shifted arrow keys erase the points and move the cursor in the same manner as the unshifted arrow. If you enter an ASCII character, it is printed on the screen. The control characters are all shifted ASCII characters. The control functions appear in Table 1.

The SHIFT I command inverts the screen contents. That is, if a graphics pixel is on, this command turns it off, and vice versa. One method of performing this is to test each graphics bit with POINT(X,Y). If a pixel is SET, then it is RESET, otherwise it is SET. This SETs all points that contain ASCII characters, and that can be a problem. Another approach, the one taken here, uses byte rather than bit resolution, and so is more than six times faster. It becomes a simple matter to leave ASCII letters alone when you test the entire byte.

The graphics blocks are arranged in a certain fashion which allows SET and RESET to work by ORing the current pixel position with the graphics byte currently in video memory. The current byte is PEEKed at, and 128 (80H) is subtracted from this amount. If the byte contains a space (32) or an

ASCII character, this difference is less than zero. You can test for a space and set it to a graphics space (128). Subtract this number from 191 to get the inverse graphics character. POKE this character into the video memory and get the next byte. This method is considerably faster than using SET and RESET and allows you to keep ASCII characters on your screen throughout the inversion process.

Up arrow	Moves cursor up and draws
Down arrow	Moves cursor down and draws
Left arrow	Moves cursor left and draws
Right arrow	Moves cursor right and draws
SHIFT X arrow	Same as normal arrow except it erases
SHIFT S 1	Saves screen in buffer 1
SHIFT S 2	Saves screen in buffer 2
SHIFT R 1	Redraws screen from buffer 1
SHIFT R 2	Redraws screen from buffer 2
CLEAR	Erases to the end of the screen
SHIFT C	Erases screen and begins over
SHIFT B	Breaks or leaves program
SHIFT I	Reverses the video screen
CHR$	Prints ASCII CHR$ on screen at cursor position

Table 1. *Control functions*

After you draw two or more screens, you may clear the screen by pressing SHIFT C, and the program redraws the screens. This part is fairly fast since byte resolution is used to redraw them. If you compile this program, it is very fast. After you get them the way you like them, break out of the program by pressing SHIFT B and load your monitor. In BASIC, of course, you can use the BREAK key, but if the program is compiled, the BREAK key does not work.

I have written this program in a slightly peculiar syntax so that it can be directly compiled using ZBASIC for additional speed and will run in BASIC as well. Be sure to change the locations of the buffers if you compile it so they do not interfere with the memory mapped variables of ZBASIC.

All the variables are defined as integers for increased speed. When you use the program, be sure to reserve high memory for the 1K screen buffers. I use two buffers so that I can design a screen and save it, then invert the screen using the SHIFT I command and save the inverted screen. I like being able to use a logo and invert it for visual effects. It is difficult to redraw the screen by hand to be the exact inverse of the previous screen; so I have the program do it for me. If you decide to have more screen buffers, be sure that you have 1K (1024 bytes decimal or 400 bytes hex) reserved. In the program below, the buffers are located for a 48K machine. Buffer 1 is located at F000H.

Remember that since PEEKs and POKEs require a 16-bit signed integer, if the address is greater than 32767, you must use the rule ADDRESS2 = ADDRESS1 – 65536. You can then use address 2 for the start of the buffer. F000H is 15 × 4096 (61440), which is too large; so you must use the rule to get the correct address (61440 – 65536 = – 4096). If you use Disk BASIC, and do not intend to compile the program, you may use hexadecimal constants which are &HF000 and &HF400 in this case.

To save a screen, press SHIFT S, and the program goes into an INKEY$ loop, looking for the buffer number. It does not prompt you because this would destroy the painstakingly drawn screen. Once the buffer is chosen, the screen bytes are read, one at a time, using PEEK, and then are POKEd into the buffer area. When control returns to you, OK appears in the upper left-hand corner, and the cursor begins to flash. Redrawing a screen uses the same INKEY$ loop to look for the buffer number and uses a similar strategy to redraw the screen.

Once you have saved the screens, load a monitor such as T-BUG or TASMON, and you can save the screen to disk or tape as a single block of memory. This practice saves a fair amount of tape or disk space as compared to using DATA statements or stored arrays, both of which require 4K of memory to store 1K of information. This is because both data statements (stored as ASCII numbers) and array data (stored as two-byte integers) take a lot of space. With four bytes per screen byte for data (three for the three digits, and one for the comma), or two bytes for each element of an integer array, these methods require a lot of memory.

Once you have designed and saved your screens, you can use them in BASIC or machine-language programs. With a BASIC program, copy the REDRAW routine from the program. It reads in the numbers via PEEK and then POKEs them into video memory, updating the counters by 1, until it has PEEKed and POKEd all 1024 bytes. In machine language, it is even easier. The Z-80 has block move commands not available on most other CPUs. In this case, you use LDIR. See Table 2.

```
100  LD    HL,BUFFER   ;where the screen is stored
110  LD    DE,3C00H    ;the video memory
120  LD    BC,400H     ;1K byte to move
130  LDIR              ;do it!!!!
140  RET               ;from whence you came
```

Table 2. *Display routine*

You can store a specially designed screen efficiently in memory and, in less than 1/40 of a second, zap it onto the screen for a flashy display. If you want to use a BASIC driver program, use the short assembly-language routine to

put it on the screen quickly with USR( ), and return to BASIC. For example, if your screen buffer starts at 7A00H, put the machine-language routine at 7994 and end it with a RET. You can then save the screen and the machine-language routine as a single unit and have lightning fast logos via USR.

Program Listing. *Screen editor (SCRNEDIT)*

Encyclopedia Loader

```
 0 REM   ******          SCRNEDIT          ******
 2 REM   ******    BY BRUCE DOUGLASS       ******
 4 REM   ******   DEPT. OF PHYSIOLOGY      ******
 6 REM   ******   SCHOOL OF MEDICINE       ******
 8 REM   ******  USD, VERMILLION, S.D.     ******
10 DEFINT A - Z:
   CLS :
   S = 15360:
   F$ = "":
   X = 64:
   Y = 24
20 GOSUB 30:
   GOSUB 180:
   POKE 16444,0:
   GOTO 20 :
   '  POKE RESETS KEYSCAN
30 A$ = INKEY$:
   IF A$ = F$
    THEN
     RETURN :
    ELSE
     A = ASC(A$)
40 IF A = 8
    THEN
     IF X > 0
      THEN
       X = X - 1:
       SET(X,Y):
       RETURN :
      ELSE
       RETURN
50 IF A = 9
    THEN
     IF X < 127
      THEN
       X = X + 1:
       SET(X,Y):
       RETURN :
      ELSE
       RETURN
60 IF A = 10
    THEN
     IF Y < 47
      THEN
       Y = Y + 1:
       SET(X,Y):
       RETURN :
      ELSE
       RETURN
70 IF A = 91
    THEN
     IF Y > 0
      THEN
       Y = Y - 1:
       SET(X,Y):
       RETURN :
      ELSE
       RETURN
80 IF A = 115
    THEN
     220 :
     '  SAVE ROUTINE
90 IF A = 114
    THEN
     280 :
     '  REDRAW ROUTINE
100 IF A = 27
    THEN
```

```
      IF Y > 0
       THEN
        Y = Y - 1:
        RESET(X,Y):
        RETURN :
       ELSE
        RETURN
110 IF A = 26
     THEN
      IF Y < 47
       THEN
        Y = Y - 1:
        RESET(X,Y):
        RETURN :
       ELSE
        RETURN
120 IF A = 24
     THEN
      IF X > 0
       THEN
        X = X - 1:
        RESET(X,Y):
        RETURN :
       ELSE
        RETURN
130 IF A = 25
     THEN
      IF X < 127
       THEN
        X = X + 1:
        RESET(X,Y):
        RETURN :
       ELSE
        RETURN
140 IF A = 98
     THEN
      STOP :
      '   <s> B FOR "BREAK"
150 IF A = 99
     THEN
      10 :
      '   <S> C FOR CLEAR SCREEN
160 IF A = 105
     THEN
      330 :
      '   <s> I FOR INVERSE VIDEO
170 P1 = X / 2 + INT(Y / 3) * 64:
    PRINT @P1,A$;:
    X = X + 2:
    IF X > 127
     THEN
      X = 0:
      IF Y < 44
       THEN
        Y = Y + 3:
        RETURN :
       ELSE
        RETURN
175 RETURN
180 REM    ******       BLINK ROUTINE       ******
190 IF POINT(X,Y) = - 1
     THEN
      RESET(X,Y):
      P = - 1:
      :
     ELSE
      SET(X,Y):
      P = 0
200 GOSUB 390:
    IF P = - 1
```

Program continued

```
     THEN
      SET(X,Y) :
     ELSE
      RESET(X,Y)
210 RETURN
220 REM    ******      SAVES SCREEN BYTES      ******
230 GOSUB 300 :
    ' GET SCREEN BUFFER LOCALE
240 FOR I = 0 TO 1023:
     POKE E + I, PEEK(S + I):
     NEXT I
250 PRINT @0,"OK";
260 RETURN
270 REM    ******      REDRAWS SCREEN      ******
280 GOSUB 300:
    FOR I = 0 TO 1023:
     POKE I + S, PEEK(E + I):
     NEXT I
290 GOTO 20
300 A$ = INKEY$:
    IF A$ = F$
     THEN
      300 :
     ELSE
      A = ASC(A$)
310 IF A = 49
     THEN
      E = - 4096 :
     ELSE
      IF A = 50
       THEN
        E = - 3072
320 S = 15360:
    RETURN :
    '  BUFFERS SET FOR OF000H AND 0F400H
330 FOR I = S TO 16383
340  A = PEEK(I) - 128:
     IF A < 0
      THEN
       IF PEEK(I) = 32
        THEN
         A = 0 :
        ELSE
         GOTO 360
350  POKE I,191 - A
360  NEXT I:
    GOTO 20
390 FOR I = 1 TO 10:
     NEXT I:
    RETURN
```

HARDWARE

Lowercase Driver for the TRS-80
Minor Monitor Maintenance

Lowercase Driver for the TRS-80

by John A. Hassell

Installing a lowercase modification on a TRS-80 Model I costs less than five dollars in parts and takes less than an hour. This article tells you how to make your computer work right with very little expense.

The character generator ROM, a special memory chip for generating the characters that appear on the screen, contains both the uppercase and lowercase characters. Radio Shack, however, did not use the chip. Cracking open the keyboard was a little too threatening at the time; so I delayed the decision. When the text-processing software Electric Pencil became available for the TRS-80 with its detailed instructions on installing a lowercase modification, I consulted an electronics expert. We opened the case and installed the lowercase modification, including the special key and a switch to turn off the modification when not needed.

The modification worked beautifully but had to be turned off when not being used with Electric Pencil. When it was on, instead of normal letters, a set of circles, arrows, squares, and other characters appeared on the screen. The computer could understand them, but I could not.

I wrote a software driver that provided both uppercase and lowercase letters, including a SHIFT lock and the use of the special characters, but I still had to turn the lowercase modification off until I loaded the driver. The irritation continued until I took a second look at Steven Wexler's article, "Lowercase for the TRS-80" in *Kilobaud Microcomputing*, April 1980, p. 132. This modification was simpler and did not require a switch to turn off the funny characters and return to the normal uppercase letters. The computer came up with normal characters when the computer was turned on. Since I was used to using a switch, I thought the modification was wrong. A quick look at the technical reference manual and a close look at Mr. Wexler's diagram in Figure 1 of his article revealed a new approach. He decodes bit 6, the bit that would be displayed by the video chip that Radio Shack left out, through a spare gate so that any time the special characters would be shown on the screen the uppercase character appears instead. The Radio Shack modification works the same way but replaces the character generator chip which replaces the special characters with uppercase characters.

Mr. Wexler's modification does not require a switch and a special key as the Electric Pencil modification does. The only problem with converting to Mr. Wexler's modification was that the RAM in the Electric Pencil

modification is piggybacked on video chip Z45 and Mr. Wexlers' modification is piggybacked on Z63. A careful look at the service manual, however, reveals that all the lines, except pins 11 and 12, to all seven video chips are the same. Consequently, you can put the piggyback RAM on any one of the seven video chips. These are the chips numbered Z45 to Z48 and Z61 to Z63 in Figure 1 and Photo 1. The two enable lines, pins 11 and 12, must remain separate.

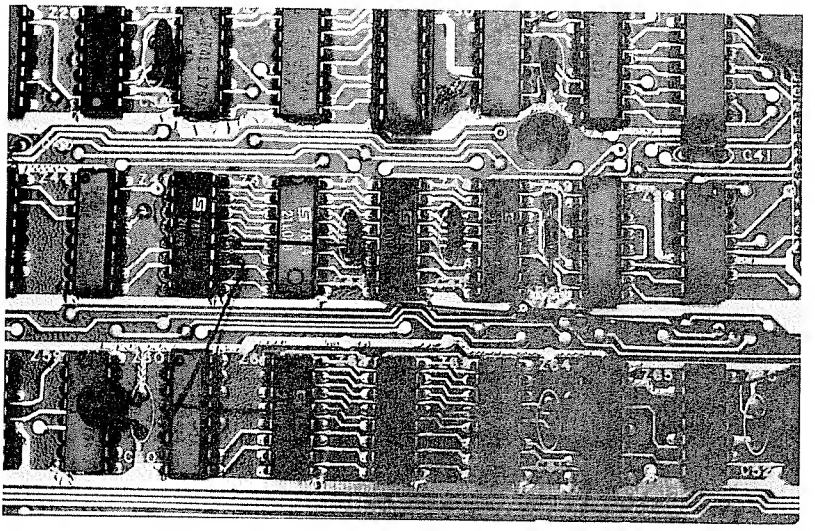


Photo 1. *Closeup of the video RAM chips. The video chips are on the second and third rows; the chip identification number is to the upper left of the chip. On this board, the piggyback is on chip Z45, the third chip from the left in the second row.*

I had to decide if the circles, broken arrows, rectangles, and the like were important. The driver I had written allowed the display of these characters as part of the full character set. With the new modification, these characters would no longer be available. The only time I used these characters was to show how neat my driver was. I had yet to include these special characters in serious programming; so I eliminated them.

Examination of the inside of the computer revealed that the one trace that must be cut had already been made with the Electric Pencil modification. The cut is the dark mark between the fifth and sixth chips in the top row under the Z30 label shown in Photo 1. I removed the switch installed for Electric Pencil and began to go through the instructions. A careful examination of the procedure revealed a few changes that can lead to a simpler and safer technique to install the modification.

1) The extra chip can reside on any one of the existing video RAMs. If you

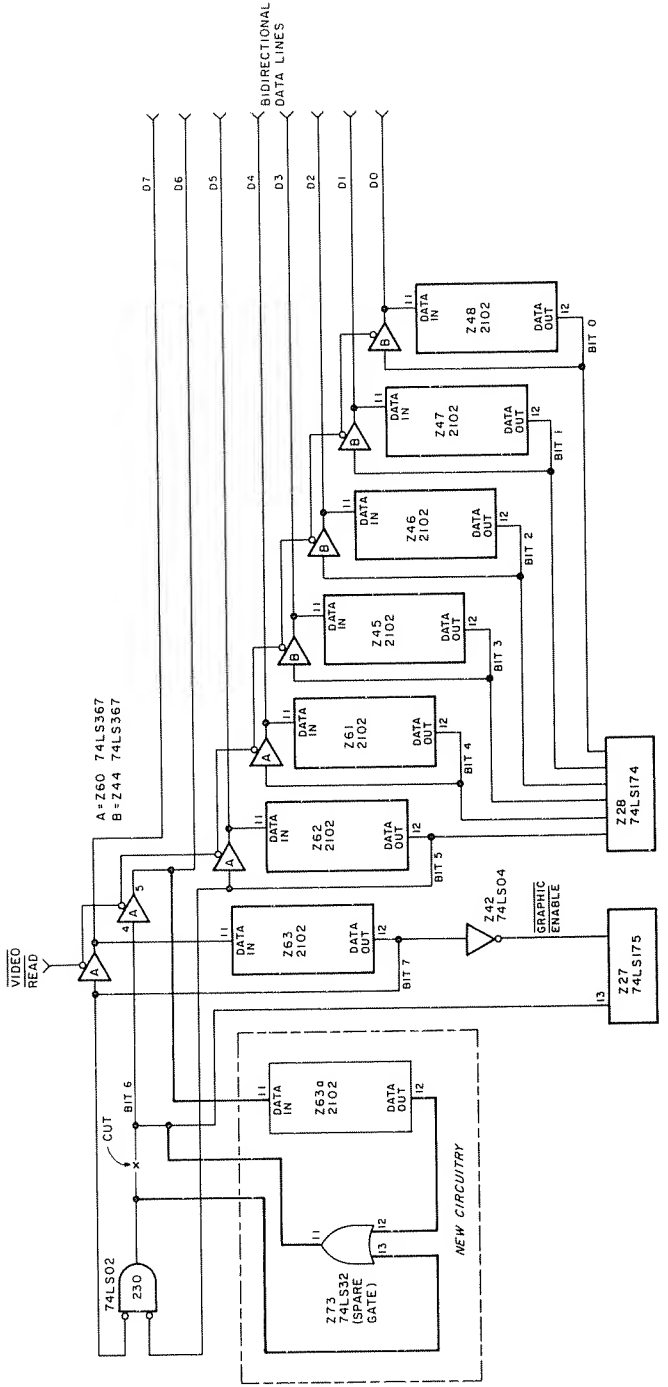


Figure 1. *Circuit diagram of video RAM and Wexler's modification (from* Kilobaud Microcomputing, *April 1980)*

have one of the other modifications in which the extra chip is on a chip other than Z63, do not remove it. Only pins 11 and 12 are important.
2) Do not try to desolder pins 11, 12, and 13 of Z73 and bend them up. This can damage the chip or break a pin. Turn the board over and cut the connecting traces between the three pins, 14, 13, and 12, by making a V-shaped cut through the foil with a knife. If the solder is too thick to make the cut, remove the excess with a solder wick and then make the cuts. The finished cuts are shown in the center of Photo 2. If you ever want to restore this modification, use a little excess solder to bridge the gap.

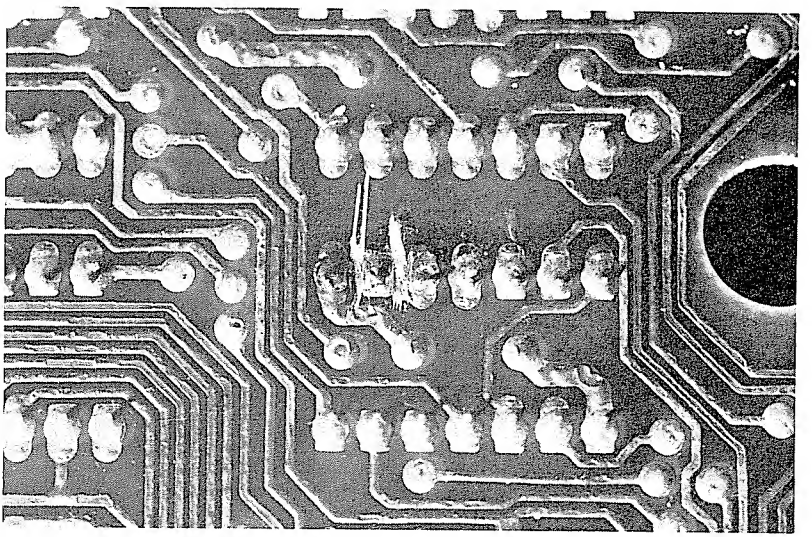


Photo 2. *Closeup showing the bottom of chip Z73 where the traces between pins 12 and 13 and pins 13 and 14 have been cut*

3) Rather than soldering posts onto the connections, solder wire-wrap wire to the pins. This requires less solder and less heat stress on the chips. Wire wrap is a small gauge insulated wire (see Photo 1) used to make wrap connections on circuit boards.
4) In running the wires between connections, put them under the capacitors laying in line with wires preventing them from getting caught or dangling free. See Photo 3.
5) Before you close up the case, carefully check all connections, connect the video and the power supply, and try the modification. MEMORY SIZE should appear on the screen. If not, turn it off and recheck all connections.

Once the keyboard is reassembled, you cannot see the display of the lowercase unless you POKE the values directly into the video memory. Use the following one-line BASIC program to test this.

```
10 CLS : FOR I = 32 TO 191 : POKE 15360 + I*4, I : NEXT
```

Neither shifted nor unshifted keys produce the display of the lowercase even though it is stored as lowercase in memory. This is a problem with Radio Shack's ROM. A patch to both the video and keyboard drivers produces a keyboard function like those of larger computers.

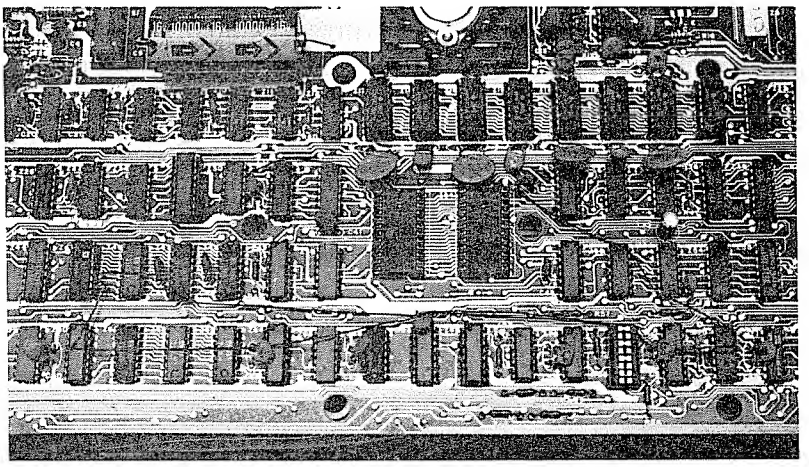


Photo 3. *The finished board with wire-wrap wire weaved under the capacitors to prevent them from dangling*

I must warn you of the consequences of installing this modification. The Electric Pencil no longer works in the lowercase mode. This, along with the scrolling problem, has been corrected by a commercial software patch. The Radio Shack lowercase modification also does not work with the Electric Pencil for the same reason. Any software that works with the Radio Shack modification works with this modification as well.

Once you have corrected the hardware, a software change is necessary before you can take advantage of it. Both the video and the keyboard must be patched before you can obtain total use. The video patch is necessary to distinguish uppercase characters from lowercase characters since ROM treats both the same for display. The keyboard needs to be patched so that the keyboard can have both unshifted lowercase with shifted uppercase, or a SHIFT lock (all uppercase). All the new disk operating systems, except Radio Shack's, include such a driver.

The source listing of an assembler program to implement the capabilities

of the lowercase modification is shown in Program Listing 1. This driver uses the keyboard decoding done by the Level II ROM and requires only 55 bytes of memory space to handle both the keyboard and the video correction routines. I provided the assembler listing for those who would like to have a fast, efficient loading program and a BASIC program for those who prefer to run from BASIC. (See Program Listing 2.) Both of these programs determine the top of usable memory and move the program to the memory location just below that value. Once the program has been moved, the top-of-memory pointer address is recalculated and put into the top-of-memory address pointers at 4049H and 40B1H. 4049H is the DOS pointer, and 40B1H is the BASIC top-of-memory pointer. The addresses of the keyboard and video are put into the appropriate driver blocks, and control returns to the sending system, either DOS or BASIC.

The SHIFT-lock function is the key combination that causes the alphabetic characters to display lowercase unshifted and uppercase shifted characters, or in the locked position uppercase characters only, whether shifted or not. This function is the control character U. The function is generated by pressing the SHIFT, down-arrow, and the U keys at the same time. No response is seen on the screen, but the effect is seen immediately when you press an alphabetic key. When the SHIFT lock is in effect, only uppercase characters are displayed, whether the SHIFT is pressed or not. When the lock is not on, lowercase characters are produced for unshifted characters and uppercase characters appear for shifted characters. This feature does not affect any keys but alphabetic keys. In addition, BASIC uses the case character that is shown on the screen or converts it if uppercase characters are required. You no longer have to hold down the SHIFT key for lowercase letters.

If you want to use a control character other than U, you can change it to any character not used by Level II. The values of the control characters and the usable characters for this function are shown in Table 1. For example, if control Z is more desirable than U, substitute the value of 1AH (26) for 15H (21) in FLCHAR. In the BASIC program, use a value of 26 instead of 21 in line 1002.

Type the BASIC program exactly as shown. You can generate the lowercase string in quotation marks in line 40 before you execute the program by holding down the SHIFT key for each character. When you run the program, the response is in lowercase characters, confirming that the driver is active. A more convenient method is to type the program, save it in case something goes wrong, run it, and then edit line 40. After running the program, push the SHIFT, down-arrow, and U keys at the same time to activate the lowercase modification. Typing any alphabetic character produces the lowercase characters. Since the ROM automatically converts to uppercase where required, you needn't worry about typing the commands

in uppercase. Type EDIT 40 and you will be in the edit mode in line 40, indicated by the 40 which shows up on the screen followed by a space and the cursor. Step through the text by pressing the space bar until you reach the first character inside the quotation marks. Now give the hack command (H), retype the text in lowercase, and save it. From now on, when the program is first loaded after the machine is turned on, the string inside the quotation marks appears as uppercase characters, but when you run the program, the screen shows "The Lowercase Driver is now working." If you list the program, you see both uppercase and lowercase text. The prompts from BASIC appear in both cases, for example, Ready.

Character (1)	Value		Level II Use
	Decimal	Hex	
A (2)	1	01H	Break
B	2	02H	None
C	3	03H	None
D	4	04H	None
E	5	05H	None
F	6	06H	None
G	7	07H	None
H (2)	8	08H	Back space
I (2)	9	09H	Tab
J (2)	10	0AH	Line feed
K	11	0BH	None
L	12	0CH	None
M (2)	13	0DH	Enter key (LF/CR)
N	14	0EH	None
O	15	0FH	None
P	16	10H	None
Q	17	11H	None
R	18	12H	None
S	19	13H	None
T	20	14H	None
U	21	15H	None
V	22	16H	None
W	23	17H	None
X (2)	24	18H	Erase line
Y (2)	25	19H	32-Character mode
Z	26	1AH	None

(1) Control characters are generated by pressing the SHIFT, down arrow, and the key in column 1 at the same time.
(2) Do not use these characters for SHIFT locks.

Table 1. *Control characters and associated values*

The CLEAR statements in the beginning and end of the program are necessary for BASIC to realign the memory size and string pointers. The value of 50 in line 40, however, is not critical. You can use any value, but the system normally comes up with a value of 50. If the assembler program is executed from Level II, a CLEAR 50 should be executed immediately after to realign the pointers, or else BASIC might get lost.

Neither of these programs interferes with programs that are located or to be located in high memory as long as you have set the HIMEM. You can set the memory size, and the driver loads itself below the size you specify. Since both listed programs use the current HIMEM value to load, then recalculate the HIMEM value, multiple executions of either program cause the memory to be reduced continuously. To avoid this problem, either reset the HIMEM value, push the reset button for DOS, or type SYSTEM followed by /0 in BASIC. This driver works in any program that uses the ROM video and keyboard calls.

Program Listing 1. *Lowercase driver, assembler listing*

```
                00100 ; ******************************************************
                00110 ;       Lowercase Driver for TRS-80 Model I
                00120 ;                    by
                00130 ;              John A. Hassell
                00140 ;******************************************************
                00150 ;       For  use with either Level II Basic or DOS
                00160 ;       *****   With Level II make RETURN = 1A19H  *****
                00170 ;          *  and use BHIMEM for HIMEM in line 210  *
                00180 ;       *****     With DOS make  RETURN = 402DH  *****
                00190 ; ******************************************************
7500            00200         ORG     7500H           ;LOAD AT A CONVIENT PLACE
7500 ED5B4940   00210 START   LD      DE,(HIMEM)      ;GET THE HIMEM VALUE
7504 013800     00220         LD      BC,ENDING-KB+1  ;PUT THE SIZE IN BC
7507 215E75     00230         LD      HL,ENDING       ;END OF PROGRAM
750A EDB8       00240         LDDR                    ;MOVE IT TO THE TOP OF
                00250 ; AVAILABLE MEMORY
                00260 ;    **** The driver is now in high memory *****
                00270 ; ******************************************************
750C ED534940   00280         LD      (HIMEM),DE      ;RESET DOS HIMEM VALUE
7510 ED53B140   00290         LD      (BHIMEM),DE     ;AND BASIC TOP OF MEMORY VALUE
7514 13         00300         INC     DE              ;INC TO KB ROUTINE ADDRESS
7515 ED531640   00310         LD      (KBBLK),DE      ;PUT ADDR IN KB BLOCK
7519 ED534940   00320         LD      (HIMEM),DE      ;RESET THE HIMEM ADDRESS
751D 211900     00330         LD      HL,VIDEO-KB     ;CALCULATE THE VIDEO
7520 19         00340         ADD     HL,DE           ;ROUTINE ADDRESS POSITION
7521 221E40     00350         LD      (VDBLK),HL      ;PUT IN VD BLOCK ADDR
7524 C32D40     00360         JP      RETURN          ;ALL DONE SO RETURN
                00370 ; ******************************************************
                00380 ; ****  Start of the Keyboard routine *****
                00390 ; ******************************************************
7527 CDE303     00400 KB      CALL    ROMDVR          ;GET KEY FROM  ROM DVR
752A 57         00410         LD      D,A             ;SAVE CHARACTER IN D
752B FE15       00420         CP      FLCHAR          ;TEST FOR THE CAPS LOCK CHAR
752D 3A1940     00430         LD      A,(FLAG)        ;AND GET THE CAPS LOCK FLAG
7530 2005       00440         JR      NZ,NOTON        ;JUMP IF NOT LOCK CHAR
7532 EE01       00450         XOR     1               ;IF ON TOGGLE THE FLAG
7534 321940     00460         LD      (FLAG),A        ;AND PUT IT BACK
7537 B7         00470 NOTON   OR      A               ;TEST IF CAPS LOCK ON
7538 7A         00480         LD      A,D             ;PUT CHARACTER BACK
7539 C8         00490         RET     Z               ;IF OFF BYPASS LOCKS
753A FE40       00500         CP      40H             ;TEST IF ALPHABETIC
753C D8         00510         RET     C               ;QUIT IF NOT
753D CBAF       00520         RES     5,A             ;MAKE CHAR A CAPITAL
753F C9         00530         RET                     ;END OF KB ROUTINE
                00540 ; ******************************************************
                00550 ; ****   Start of Video Routine    ****
                00560 ; ******************************************************
7540 DD6E03     00570 VIDEO   LD      L,(IX+03H)      ;GET THE CURRENT SCREEN
7543 DD6604     00580         LD      H,(IX+04H)      ;ADDRESS INTO HL REG
7546 DA9A04     00590         JP      C,049AH         ;TEST FROM ROM VALUE
7549 DD7E05     00600         LD      A,(IX+05H)      ;GET CHARACTER
754C B7         00610         OR      A               ;IS IT NULL
754D 2801       00620         JR      Z,BPASS         ;IF NOT NULL JUMP
754F 77         00630         LD      (HL),A          ;PUT CHAR TO SCREEN
7550 79         00640 BPASS   LD      A,C             ;RESTORE CHARACTER
7551 FE20       00650         CP      20H             ;IS IT CONTROL CHAR
7553 DA0605     00660         JP      C,0506H         ;GO TO CONTROL AREA IN ROM
7556 FE80       00670         CP      80H             ;IS IT GRAPHIC
7558 D2A604     00680         JP      NC,04A6H        ;GO TO GRAPHIC AREA IN ROM
755B C37D04     00690         JP      47DH            ;GO TO NORMAL CHAR RETURN
755E 00         00700 ENDING  DEFB    00
0015            00710 FLCHAR  EQU     0015H           ;CONTROL U CHARACTER
03E3            00720 ROMDVR  EQU     03E3H           ;ADDR OF ROM DVR
4016            00730 KBBLK   EQU     4016H           ;KB DRIVER BLOCK ADDRESS
4019            00740 FLAG    EQU     4019H           ;FLAG IN KB BLOCK
401E            00750 VDBLK   EQU     401EH           ;VD BLOCK ADDRESS
4049            00760 HIMEM   EQU     4049H           ;DOS HI MEMORY
40B1            00770 BHIMEM  EQU     40B1H           ;BASIC  HI MEMORY
402D            00780 RETURN  EQU     402DH           ;DISK RETURN
                00790 ;RETURN EQU     1A19H           ;THE BASIC RETURN
7500            00800         END     START
00000 TOTAL ERRORS
```

Program continued

Program Listing. *Lowercase driver, BASIC listing.*

```
10 CLEAR 1000:DEFINT I,J
20 FORI=0TO86:READI1:POKE&H7500+I,I1:NEXT
30 DEFUSR=&H7500:I=USR(0)
40 CLEAR50:PRINT"The Lowercase Driver is now working"
1000 DATA237,91,177,64,1,56,0,33,87,117,237,184,19,237,83,22
1001 DATA64,27,237,83,177,64,33,26,0,25,34,30,64,201
1002 DATA0,0,205,227,3,87,254,21,58,25,64,32,5,238,1
1003 DATA50,25,64,183,122,200,254,64,216,203,175,201
1004 DATA221,110,3,221,102,4,218,154,4,221,126,5,183
1005 DATA40,1,119,121,254,32,218,6,5,254,128,210,166
1006 DATA4,195,125,4,0
```

Minor Monitor Maintenance

by Nick Doble

There have been a number of articles about making electronic modifications to the TRS-80 Model I monitor. This article may also apply to the Model III monitor, but I am not familiar with it. Not everyone has the interest, knowledge, or ability to make these modifications, yet there is some minor monitor maintenance that is useful and that anyone can perform. Everyone is familiar with vertical and horizontal hold and the other common monitor adjustments, but there are three other adjustments that can readily be made to the display on the CRT (cathode ray tube) of the monitor: tilt, centering, and vertical size.

When you work with the monitor, you must be careful not to touch the chassis with your hand or a tool while making the adjustments because the AC line is connected directly to the metal chassis of the monitor. The possibility of getting a shock is discussed in more detail at the end of this article. When you work with high voltages, it is a good idea to use only one hand when making adjustments, so as not to provide a path for electricity through the body.

To make the adjustments to your monitor, you need some kind of graphics to help you align the display. The Program Listing is a short and simple Level II or Disk BASIC program for drawing a series of "Chinese rectangles" on the face of the CRT (see Photo 1)—this is a better test display than a simple white-out, because a white-out tends to distort the display. After you have entered and saved the program, open the back of the monitor by removing the four screws at the corners, and the fifth next to the power cord. You need a 1/4 inch socket driver to remove these screws. You might want to unplug the monitor to do this; you may also want to detach the back from the line cord so that it doesn't get in the way.

The most difficult aspect of this project is reinstalling the back of the monitor when you are finished making the adjustments. The wells that hold the screws for the back are not closed, and the screws don't seem to have any desire to go where they should. You will have to persevere.

Adjusting the Tilt of the Display

The tilt of the display on the face of the CRT is governed by the position of the yoke on the neck of the CRT. As you look at the open back of the monitor (see Photo 2), you see that the CRT is Y shaped. The upper part of the Y is the

face of the tube, and the bottom half is the neck. There is a device shaped like a basket or a yoke on the tube where the neck begins to expand into the face of the CRT. On my monitor this yoke is white. There should be a handle on either side of the yoke. These handles are used to turn the yoke left or right to adjust the tilt of the picture on the face of the tube.

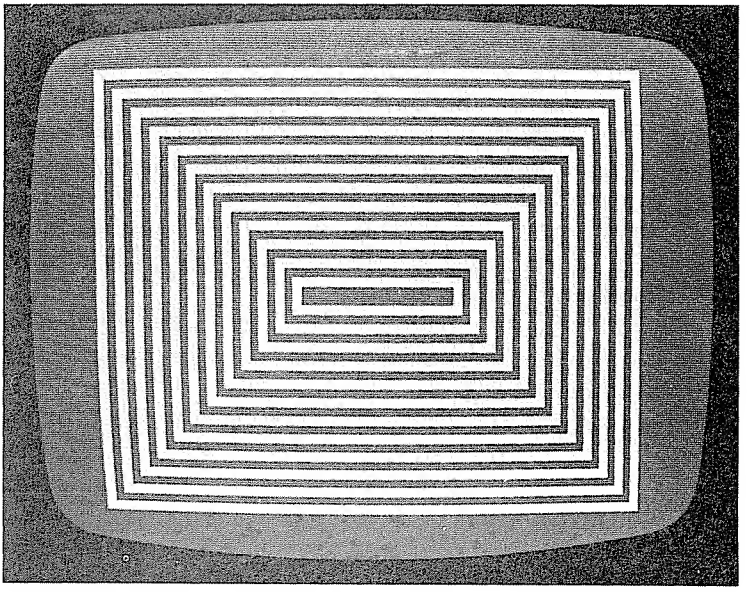


Photo 1. *"Chinese rectangles" to help you align the CRT display*

There is also a metal band toward the back of the yoke, with a screw on it to tighten or loosen the band (again, see Photo 2). In making the following adjustments, if you are unable to move the yoke, loosen this screw. When you've finished, tighten it again, but only as much as you loosened it, or you may put too much strain on the neck of the CRT. If the yoke still refuses to move after you have loosened the band, apply a little circular force to the yoke—it may be stuck to the neck of the tube from heat and the pressure of the band. Be careful, as the neck of the CRT is very fragile; do not grab the neck with your hand or apply other than circular force to it.

With the Chinese rectangle program displayed on the CRT, adjust the yoke to the left or right so that the display is parallel with the sides of the CRT. Now retighten that screw if you loosened it. That completes the tilt adjustment.

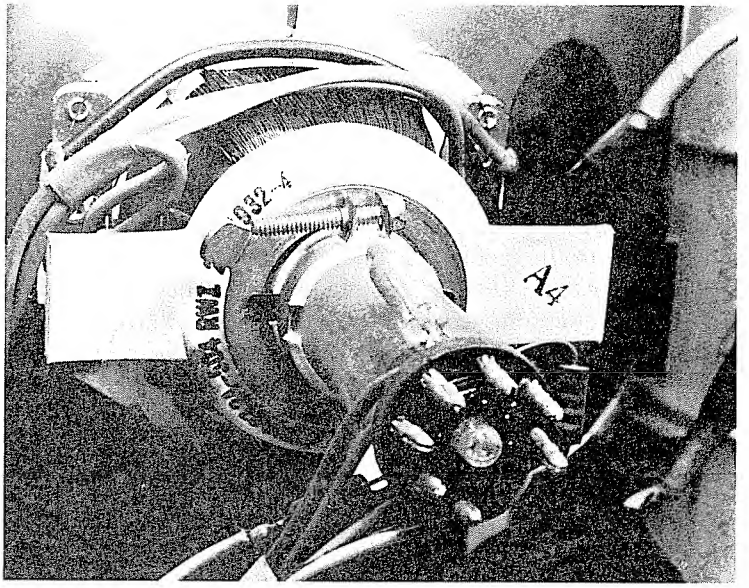


Photo 2. *Open back of the monitor*

How to Adjust the Centering

The spaces between the top and bottom of the graphics display and the top and bottom of the CRT should be about the same; and the spaces between the sides of the display and the two sides of the CRT should also be about the same, as shown in Photo 1. If they are not, you need to adjust the centering of the display on the screen. Notice two rings with a tab on each at the back of the yoke (see Photo 2). These rings are permanent magnets, and their positions around the electron beam being generated in the neck of the CRT influence where the beam is aimed on the face of the CRT, just as the wind might move water from a hose. The tabs are there for you to hold on to, so move the centering rings around the neck, individually and together, and see what effect they have on the positioning of the graphics display. After you have the hang of how they influence the display, center the display so there are equal spaces top and bottom, and side to side.

The Vertical Height Adjustment

The remaining adjustment is for vertical height. Looking at the back of the monitor, you see a circuit board standing on its side toward the left of the monitor; slightly to the right of this board and near the front (actually the back of the monitor) of the big board that the upright board is plugged into,

you see what is called a trim pot. On my monitor, this control is blue, and VERT. SIZE is printed on the PC board in front of it (see Photo 3). There is a slot in the front of the pot, and using a screwdriver in this slot, you can turn the pot to the left or right while watching the face of the CRT. This is the time to be careful of a shock, as you are near the chassis, with your eyes on something else. Some people like to use a mirror to watch the CRT display. You will see the picture expand and shrink vertically as you move the control. It will be obvious where the live part of the picture ends; adjust it so that the edge of the picture is at the edge of the tube. You will probably find that the two edges of the picture do not touch the top and bottom edges of the tube at the same time, so adjust for whichever touches last. Now expand the picture so it overlaps the edges of the CRT by about 1/2 inch; you will have to guess about this because you can't see it. This is necessary to make sure that changes in line voltage, which affect the size of the picture, do not shrink it so much that the picture's edges become visible.

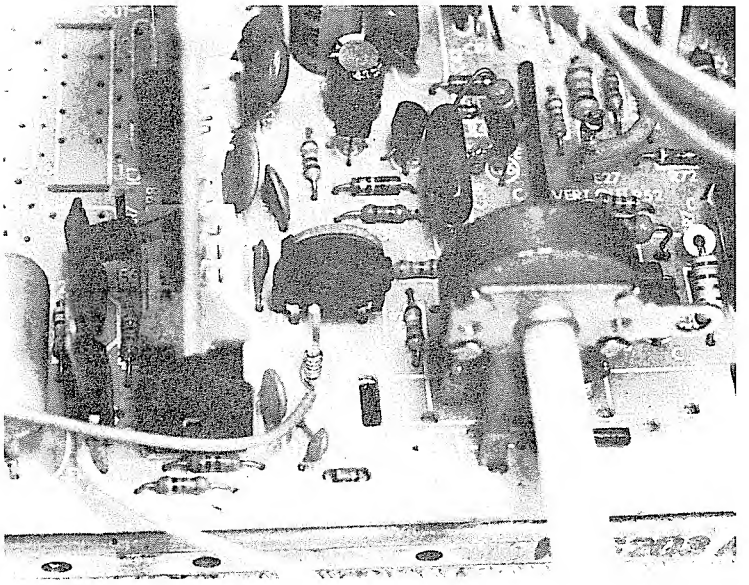


Photo 3. *Vertical size adjustment*

You have now finished the adjustments to your monitor, and you should have a display that is parallel with the edges of the CRT and equally spaced from side to side and from top to bottom, with perhaps an inch or so from the corners of the graphics display to the top and bottom edges of the CRT,

as shown in Photo 1. The centering and height controls are somewhat interactive, so you may want to check your adjustments by going through them one more time. As your monitor ages, or the adjustments change for any reason, you can provide this routine maintenance yourself.

Working with High Voltages

In making the adjustments described in this article, you will be exposed to AC line voltage and the high voltage of the monitor picture tube. To the best of my knowledge and in my experience, AC line voltage is not lethal, despite popular belief. The severity of a shock depends on two things, current and voltage. Current without sufficient voltage is harmless, as evidenced by your 12 volt automobile battery, which has hundreds of amps in store but can't give you a shock because of low voltage. Voltage without current is equally harmless, at least to human beings, although your normal CMOS IC might be alarmed. As evidence that voltage without current won't hurt you, consider the fact that the everyday electrostatic shock has a potential of tens of thousands of volts but no current.

Despite its great current potential, under normal circumstances, 120 volts of AC is not a high enough voltage potential to break through the high resistance of the body to the extent that the current becomes harmful. Unless you insist on standing in a well grounded metal tub of dirty water (it is the impurities in water that are conductors; distilled water is an insulator) while you work on your monitor with wet hands, you are not likely to be greatly offended by any shock you may receive, if you receive any at all. Nevertheless, be careful; be sure to stand on an insulating surface such as a rug or a wood floor.

As for the high voltage to the CRT or picture tube, this is deliberately designed to be voltage without current. If you have ever cleaned the face of a TV and received a shock, you have been shocked by the high voltage on your picture tube, which is somewhere between 20 and 30 thousand volts. So you can see that there is nothing to fear; nevertheless, be careful while working with the yoke and centering rings of the monitor CRT—there is some middling high voltage at the base of the tube, and the very high voltage goes to the right side of the tube, looking from the back, as shown in the background in Photo 2.

It is probably just as well that popular belief holds that AC line voltage is lethal. I am not an expert on the subject, and there maybe other factors such as a weak heart to take into account, but I have been shocked by voltages higher than those you are working with and have not been hurt. Nevertheless, to repeat for the third time, be careful. If you are still concerned, wear a pair of rubber dishwashing gloves while working on the monitor; they are easy to wear and should eliminate any possibility of a shock.

Program Listing. *Test pattern*

```
  1 :
    '   TEST PATTERN
  2 :
    '   FOR MINOR MONITOR MAINTENANCE
  3 :
    '   BY NICK DOBLE
  4 :
    '   9/10/80
  5 :
    '
 10 CLS :
    DEFINT A - Z:
    X1 = 0:
    X2 = 127:
    Y1 = 0:
    Y2 = 47
100 FOR X = X3 TO X2:
     SET(X,Y1):
     NEXT
110 FOR Y = Y1 TO Y2:
     SET(X2,Y):
     SET(X2 - 1,Y):
     NEXT
120 FOR X = X2 TO X1 STEP - 1:
     SET(X,Y2):
     NEXT
130 FOR Y = Y2 TO Y1 STEP - 1:
     SET(X1,Y):
     SET(X1 + 1,Y):
     NEXT
140 X1 = X1 + 4:
    X2 = X2 - 4:
    X3 = X1
150 Y1 = Y1 + 2:
    Y2 = Y2 - 2:
    IF Y1 > 23
     THEN
      1000
200 GOTO 100
1000 K$ = INKEY$:
    IF K$ = ""
     THEN
      1000 :
     ELSE
      RUN
```

HOME APPLICATIONS

Can You Find that Slide?
Controlling Your Home with Your TRS-80

Can You Find that Slide?

by Robert E. Averill

After a vacation, how do you find one particular slide out of the several hundred you took on the trip? I have over 5000 slides in my collection. I number them in the upper left-hand corner with no particular regard to the contents of each slide. I number them by hand and when I get tired of numbering, I start entering the information about the slides into the data lines of the program. I can enter about 20 slides in a minute.

The program is written in Level II BASIC. You can change any slide in the program at any time without affecting the rest of the slides. When you want to remove a slide from your file for use, to send it off, or otherwise make it temporarily unavailable, reenter the data statement for that slide and add a REM statement, as in line 183, to indicate where the slide is.

Slides are coded and sorted on any or all of three vertical lines of information with up to 98 conditions in each of those lines. (See Figure 1.) This gives a total of approximately 941,192 possible combinations of descriptions for slides if you choose to code them in this manner. The number 99 is reserved in each line as a DO NOT SORT instruction to shorten search time if you can find the slide or group of slides you want without some particular line of sorting.

Where Shot	Who/What	Activity
1 OUR HOUSE	OURSELVES	
2 OTHERS HOUSE	IMMED. FAM.	BIRTHDAYS
3 OUTSIDE H.	RELATIVES	ANNIV.-OTHER
4	FRIENDS	HOLIDAYS
5 LOCAL OUT	GROUPS 1-3	MEETINGS
6 LOCAL WASH.	CROWD 3+	HISTORY FAM.
7 U.S. NOT WA.		SCENIC
8 CANADA		
9 HAWAII		BOATING

Figure 1. *Slide sort coding chart*

I enter some slides on more than one data line which allows sorting on more than one kind of search. The slide described in lines 188 and 189 was taken on a canoe trip on Bowron Lakes; therefore, coding is in line 31 column 1 which is the code for photographs taken on the Bowron trip. The photograph is of a large cow moose and her calf close to the canoe; therefore, coding for column 2 is from line 12 or 13, depending on whether I look for it

under large wild animals or small wild animals. Column 3 is coded 10 for both slides because I took the photograph while canoeing. I can find the same slide number by sorting on 31,12,10 or 31,13,10.

This slide also comes up under any sort coded 99,99,10 which brings up all slides shot while canoeing; 99,12,99 which brings up all slides containing large wild animals; 99,13,99 which is for all slides containing small wild animals; and on 31,99,99 which covers all slides taken on the Bowron Lake trip.

Set up your slide coding chart to fit your needs. Make several copies of the chart and keep one with your tapes and another with your slides in case you discover that you have a great number of slides coded, numbered, and filed by number and cannot remember the coding. Load the program down to and including line 170, then load lines 9999 and 10000. Add the data about your slides starting with line 180. There must be four pieces of data on each line. The data numbers must be separated by a comma, and the last data number must always be your slide number. If you do not want to use all of the columns in your coding, insert a 0 in its place so that there are always four pieces of information. After entering 10 or 15 slides in your program, check your work by entering RUN to be certain that you are entering the information as the program requires.

Fancier programming could make this program sort faster, but it would be considerably more complicated. This program sorts 500 slides in approximately 1 1/2 minutes. I have my slides filed in drawers of 500 with a separate program for each drawer.

Program Listing

```
 10 CLS :
    :
    '       *        SEARCH PROGRAM        *
 20 :
    ' INPUT 4 DATA FIELDS, SEPARATED BY COMMAS, SORT ON FIRST THREE
 30 :
    ' FIELDS & PRINTS NUMBERS OF 4 TH. - INPUT 99 IN FIELDS YOU DO
 40 :
    ' NOT WANT SEARCHED.
 50 CLS
 60 INPUT "ENTER CODE FOR COLUMN #1, IF NO ENTRY TYPE 99";X
 70 INPUT "ENTER CODE FOR COLUMN #2, IF NO ENTRY TYPE 99";Y
 80 INPUT "ENTER CODE FOR COLUMN #3, IF NO ENTRY TYPE 99";Z
 90 ON ERROR GOTO 9999
 95 A$ = "-"
100 READ A(1),B(1),C(1),D(1)
110 IF X = A(1) AND Y = B(1) AND Z = C(1)
     THEN
      150 :
     ELSE
      120
120 IF X = A(1) AND Y = 99 AND Z = 99
     THEN
      150 :
     ELSE
      130
130 IF X = 99 AND Y = B(1) AND Z = 99
     THEN
      150 :
     ELSE
      140
140 IF X = 99 AND Y = 99 AND Z = C(1)
     THEN
      150 :
     ELSE
      142
142 IF X = A(1) AND Y = B(1) AND Z = 99
     THEN
      150 :
     ELSE
      144
144 IF X = A(1) AND Y = 99 AND Z = C(1)
     THEN
      150 :
     ELSE
      146
146 IF X = 99 AND Y = B(1) AND Z = C(1)
     THEN
      150 :
     ELSE
      100
150 PRINT D(1);A$;:
    GOTO 100
160 :
    ' DATA ENTRY LINES #180 TO #9998 INCLUSIVE, SAMPLE BELOW
170 :
    '          SAMPLE DATA ENTRY  >180 XX,XX,XX,XXXXX
180 DATA 2,3,5,26
181 DATA 3,19,7,27
182 DATA 2,3,5,28
183 DATA 26,21,7,29:
    REM  AT M.R. STUDIO - BACK 11/15/81
184 DATA 26,12,7,29
185 DATA 2,3,5,30
186 DATA 2,3,5,31
187 DATA 5,3,19,32
188 DATA 31,12,10,33
```

Program continued

```
  189 DATA 31,13,10,33
  190 DATA 7,3,5,34
  191 DATA 6,2,13,35
  192 DATA 6,1,11,36
  193 DATA 6,28,7,37
 9999 PRINT
10000 PRINT "SEARCH IS ENDED, IF YOU WANT ANOTHER SEARCH, TYPE 'RUN' &
       ENTER":
      END
```

Controlling Your Home with Your TRS-80

Vardeman G. Moore

The TRS-80 Plug'n Power Controller is a device that allows you to control Radio Shack's Plug'n Power or BSR's System X-10 modules. These modules are remote control devices which can turn on or turn off most 110-volt home appliances and can even dim incandescent lamps. The system uses house wiring to carry control signals.

The Plug'n Power Controller, catalog number 26-1182, costs $39.95. The TRS-80 Model III and the Color Computer use the same controller as the Model I. The Plug'n Power Controller consists of a 4 1/2-inch by 4 1/2-inch by 2-inch gray box that connects to the cassette port of the TRS-80. I also received software on cassette to operate the controller with any Model I, Model III, or Color Computer. Although the controller connects to the cassette port, you can use a cassette recorder, because the controller has a socket for your recorder and a switch that allows you to change between recorder and controller operations. The controller has its own power cord which should not be plugged into your line filter if you are using one. The signals will not pass through the filter.

To use the Plug'n Power Controller, connect it and load the appropriate program for your computer following the step-by-step instructions in the manual. Then run the program. You are asked for the current time in 24-hour, military format. If nothing happens when you enter the time, make sure you have set the switch to control after using the cassette recorder. Then enter the Direct Command Mode of the controller program which is the less useful of the two modes available.

The Direct Command Mode allows you to give commands from the keyboard that are executed immediately by one of the control modules. The first command you enter is a house code (one of the letters A through P), which you can change at any time. For each house code, you get a Unit Status screen. You can enter the whole house codes: CLR to turn off all modules with the current house code, and ALL to turn on all the lamp dimmer and wall switch modules (but not appliance modules) with the current house code. Note that all of the unit codes, including those for appliances, go to ON with the command ALL. This reflects a major limitation of the BSR controller—it is a one-way system. Signals are sent, but there is no way to tell if they are received or executed. Thus, the Unit Status screen reflects the last signal sent, not the true status of the unit. This lack of verification limits

the BSR system for use at home or for a business in which an unexecuted signal does not cause a problem.

In addition to the whole-house commands, you can select an individual module in the Direct Command Mode by entering a unit code between 1 and 16 (assuming that the current house code is appropriate), then entering a command. The appropriate commands are ON and OFF. In addition, you can control the dimmer function of the lamp dimmer and wall switch modules connected to incandescent lights using the codes DIMn and BRn. These codes dim or brighten a lamp by n units on a 10-unit scale.

If you enter an invalid command in the Direct Command Mode, both the Model I and Model III generate a HELP list of the correct commands (without any explanations). The command HELP does the same thing. On Model I and Model III Level II computers, the command Q allows you to exit to BASIC or TRSDOS. Entering @ on a Model I, Model III, or Color Computer switches you from the Direct Command Mode to the Programming Mode (or back to the Direct Command Mode if you are in the Programming Mode).

The Programming Mode allows you to program a list of commands to run in the future at predetermined times. Some possible business applications are dimming the lights in a restaurant or turning on the lights and cash registers in a store in the morning and turning them off at night. At home, you can turn lights, televisions, and radios on and off in an arrangement realistic enough to make any burglar think you are at home. My wife occasionally leaves her iron on, so I have set up my system to turn the iron off late every night.

The commands you use can be any of the unit commands available in the Direct Command Mode. The Programming Mode includes a mini-editor that allows you to insert, replace, and delete lines, and to designate a line as the next one for the computer to execute. You can save programs on disk using the DUMP command.

The Programming Mode allows you to do different things on different days as illustrated in Figure 1. This feature is great for fooling your local burglar. Note in Figure 1 the order in which you must enter the data.

On the first day, light 1 turns on at 7 a.m. The television that is on appliance module 7 comes on at 5 p.m. At 10 p.m., the television goes off, followed about a second later by light 1. Line 5 is next to be executed at 19:00 on the second day. Since I don't have an A-1 module in my house, line 5 is a dummy command, and nothing happens on the second day. On the third day, the program loops back and repeats the first day's routine.

The Programming Mode allows programs of 45 lines for Model I or Model III Level II computers, 32 lines for Level I computers, and 30 lines for the Color Computer. If that isn't enough, on Model I or III Level II machines you can run a BASIC program which sends control signals as a subroutine.

#	Time	House Code	Unit Code	Command
01	07:00	M	01	ON
02	17:00	M	07	ON
03	22:00	M	07	OFF
04	22:00	M	01	OFF
05	19:00	A	01	OFF

Figure 1

The manual gives two different styles of sample programs that use BASIC and USR calls. Unfortunately, the clock in the Plug'n Power Controller doesn't work when the computer is in BASIC; so you must get the time using the TIME$ function if you have it, or use timing loops if you do not. The program in the Program Listing performs the same operations as in Figure 1. It is written for a 32K disk machine. To use this program, you must load the Radio Shack controller program, reset the computer, then enter BASIC with a memory size of 45055. Addresses for other Level II and disk machines are given in the controller manual.

One limitation of the system is that, in homes with a divided circuit box, the signals do not pass reliably from a controller on one side of the circuit box to modules on the other. If most of the things that you want to control are located on the opposite side of the circuit board from you computer, you can plug the controller in using extension cords that go to an outlet on the correct side. If you are worried about leaving the computer on for long periods to run the controller, turn off the video display and disk drives. I have left the controller, keyboard, and expansion interface on for long periods without any problems. Remember that while the computer is running the controller it cannot be used for other purposes. If you live in an apartment, your controller may affect your neighbor's circuits if a controller is in use in the next apartment.

Program Listing. *Control program, 32K or 48K*

```
  1 REM  CONTROL/BAS FOR 32K OR 48K RAM DISK
 10 REM  1. LOAD RADIO SHACK CONTROL PROGRAM        2. RESET COMPUTE
    R AND SET TIME        3. ENTER BASIC, SETTING MEMORY TO 45055
         4. RUN THIS PROGRAM WITH YOUR DATA STATEMENTS
 20 DEF USR1 = &HB100:
    REM  THIS DEFINES THE STARTING POINT OF THE
       SECTION OF THE RADIO SHACK PROGRAM WE
    WANT TO USE
 30 READ T$:
    IF T$ = "END"
     THEN
      RESTORE :
      GOTO 30
 40 A$ = MID$( TIME$,10,5)
 50 IF T$ < > A$
     THEN
      40
 60 READ HOUSE$, UNIT$, TASK$
 70 TRANSMIT$ = HOUSE$:
    GOSUB 1000
 80 IF UNIT$ < > ""
     THEN
      TRANSMIT$ = UNIT$:
      GOSUB 1000
 90 TRANSMIT$ = TASK$:
    GOSUB 1000
100 GOTO 30
1000 REM  TRANSMISSION SUBROUTINE
1010 REM  THIS SUBROUTINE POKES TRANSMIT$ INTO THE APPROPRIATE
       PLACE IN MEMORY THEN CALLS THE RADIO SHACK CONTROLLER
     PROGRAM AS A SUBROUTINE
1020 X = - 17116:
     POKE X, LEN(TRANSMIT$)
1030 FOR I = 1 TO LEN(TRANSMIT$)
1040  X = X + 1:
      POKE X, ASC( MID$(TRANSMIT$,I,1))
1050  NEXT I
1060 X = USR1(0):
     REM  THIS STATEMENT ACTIVATES THE RADIO SHACK
        ROUTINE DECLARED AS USR1 AT BEGINNING OF
     THE MAIN PROGRAM
1070 RETURN
2000 REM  DATA STATEMENTS ---- TIME SHOULD BE IN QUOTES, FORMAT
                                                        "HH:MM"
2010 DATA "07:00",M,1,ON
2020 DATA "17:00",M,7,ON
2030 DATA "22:00",M,7,OFF
2040 DATA "22:00",M,1,OFF
2050 DATA "19:00",A,1,OFF
2060 DATA END
```

INTERFACE

Speak for Yourself: A Speech Synthesizer for the TRS-80

Speak for Yourself: A Speech Synthesizer for the TRS-80

by David Hucaby

If you've ever found yourself standing in a video arcade or in your local Radio Shack store dumbfounded by those fascinating talking computers, you're not alone. If you have looked into the prospect of having one of your own, the price and complexity have probably cooled your excitement. I patiently waited until I saw the perfect chip introduced—the Votrax SC-01. With the SC-01, you can build your own voice synthesizer for a little over $72!

Votrax designed the SC-01 to produce an infinite vocabulary with a small amount of memory. Most speech synthesizers are based on the reproduction of an actual digitized human voice. The SC-01, however, creates its own sounds from within, freeing the user from great stockpiles of speech data available only in ROM. To dispose of the data requirement (the ROM chip sets), Votrax uses a technique called phoneme stringing.

Speech Basics

The SC-01 phoneme-stringing method connects individual sounds, or phonemes, of the spoken word in sequence. Votrax uses a six-bit input address to select one of 64 ($2^6 = 64$) standard phonemes, grouped as three silent sounds, 45 spoken sounds, and 16 alternative sounds, differing only in their time duration. Table 1 lists the standard Votrax phonemes by decimal code, symbol, and duration, and gives an example of pronunciation. A word such as *cat* requires only three basic sounds, and from the table can be converted into the phonemes *K-AE-T*, codes 25, 46, and 42.

With several of the phonemes, transition problems result, making the speech hard to understand. Votrax took care of this problem by using an auto-inflection function at the ends of the sounds. As a result, abrupt endings are built into *p* and *t* sounds, while ones like *sh* and *m* are given gradual transitions.

Most less expensive synthesizers feature only mechanical, monotone speech. The SC-01, however, allows its pitch to be changed with software over a short range, making it possible to give the computer personality. Using the pitch function, the synthesizer can give a command, ask a question, and even sound as if it's pouting. Two pitch input bits give a total of four different levels that you can select and change simultaneously with the phoneme codes. Consequently, you can change the pitch with each separate phoneme, allowing great versatility in expression.

Phoneme Code	Phoneme Symbol	Duration (ms)	Example Word	Phoneme Code	Phoneme Symbol	Duration (ms)	Example Word
00	EH3	59	jacket	32	A	185	day
01	EH2	71	enlist	33	AY	65	day
02	EH1	121	heavy	34	Y1	80	yard
03	PA0	47	no sound	35	UH3	47	mission
04	DT	71	butter	36	AH	250	mop
05	A2	71	made	37	P	103	past
06	A1	103	made	38	O	185	cold
07	ZH	90	azure	39	I	185	pin
08	AH2	71	honest	40	U	185	move
09	I3	55	inhibit	41	Y	103	any
10	I2	121	inhibit	42	T	71	tap
11	I1	80	inhibit	43	R	90	red
12	M	103	mat	44	E	185	meet
13	N	80	sun	45	W	80	win
14	B	71	bag	46	AE	185	dad
15	V	71	van	47	AE1	103	after
16	CH	71	chip	48	AW2	90	salty
17	SH	121	shop	49	UH2	71	about
18	Z	71	zoo	50	UH1	103	uncle
19	AW1	146	lawful	51	UH	185	cup
20	NG	121	thing	52	O2	80	for
21	AH1	146	father	53	O1	121	aboard
22	OO1	103	looking	54	IU	59	you
23	OO	185	book	55	U1	90	you
24	L	103	land	56	THV	80	the
25	K	80	trick	57	TH	71	thin
26	J	47	judge	58	ER	146	bird
27	H	71	hello	59	EH	185	get
28	G	71	get	60	E1	121	be
29	F	103	fast	61	AW	250	call
30	D	55	paid	62	PA1	185	no sound
31	S	90	pass	63	STOP	47	no sound

Table 1. *Phoneme chart*

Timing the phonemes is perhaps the most important part of obtaining good speech. Even with one sound, time duration makes for a more human quality. For example, the word *inhibit* uses three different lengths of the *i* sound. From Table 1, the three Is, on the average, last 55,121, and 80 milliseconds respectively. Otherwise, the speech sounds dull and monotonous and loses its personality.

The SC-01 contains a complete phoneme timing section which gives foolproof advice on the time duration for any sound selected. A data acknowledge/request line gives a receipt of the phoneme code sent and then signals that it is ready for another code after the prescribed amount of time

has elapsed. The sound itself, however, does not shut off; the computer must send another data code to go on to the next phoneme.

Talking with an 80

By the time I received my SC-01, I had done most of the planning for the interface to my TRS-80. When I breadboarded my first version, however, I expected the built-in phoneme timer to turn off the sound after the duration time was up. Instead, I panicked through several minutes of *aaaaaaah* while I tried to shut the thing up. Contrary to what I thought, the internal timer only informs the host computer that the sound is done and that it should send another phoneme.

In the schematic shown in Figure 1, the address lines are decoded by the 74LS04 hex inverter and the 74LS30 NAND gate so that the synthesizer responds to a port 3 address. This gives a simple address number and keeps the chip count down. If a different port number is required, simply delete the appropriate inverters in the 74LS04 until the 74LS30 output goes low at the correct address inputs. The NAND gate output is NORed with the OUT data line from the computer so that the SC-01 accepts only outgoing data. The 74LS02 output triggers the synthesizer data strobe input, which starts the timer with the audio output.

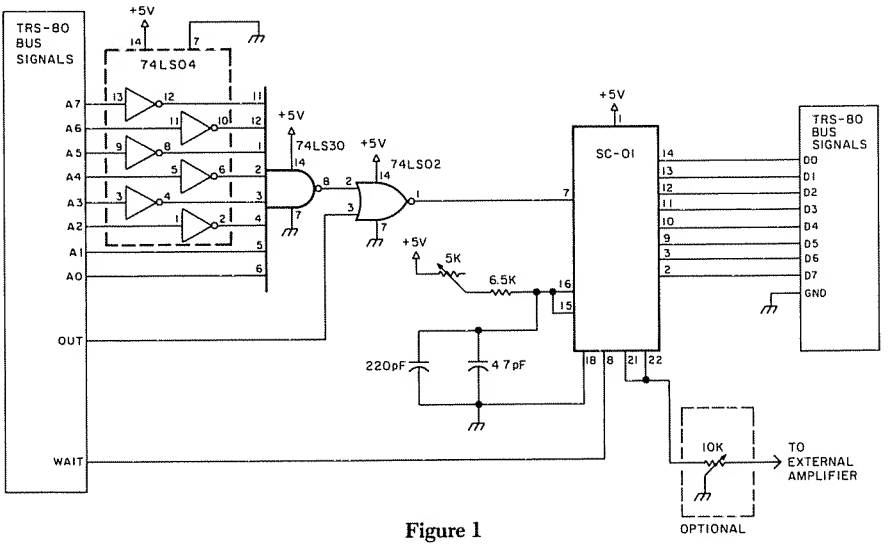


Figure 1

To achieve smooth speech from the built-in timing, I decided to make use of the RS-232 WAIT line from the TRS-80. To do this, the SC-01 simply has to stop everything in the computer while the phoneme is voiced. Then, when it is ready for another phoneme, its request line releases the WAIT hold, and

normal operations resume. Of course, this prevents the computer from operating while the SC-01 is talking, but I've found that there's not much it should have to do then anyway. Regardless, the longest timed phoneme only lasts one fourth of a second until computing power is returned.

According to the Votrax spec sheet, the SC-01 is supposed to have a normal clock frequency of around 720 kHz. This is achieved either by using a resistor/capacitor combination with its internal clock oscillator or by injecting an external clock pulse. Adjusting the 5k trim pot at pins 15 and 16 of the SC-01 gives a good range of voices by actually varying the total pitch range. Voices range from the Addams Family's Lurch to the little Martian of the Bugs Bunny cartoons. (It helps to hear it in person.) Beyond my circuit's highest voice pitch, an external clock has to be used, especially if you want to get the special effects of high-speed speech or high-frequency sounds. As you guessed, the voice speed varies directly with the pitch with any type of clock.

In my synthesizer circuit, the first six data lines of the eight from the computer board are used to select a phoneme code between 0 and 63. The most

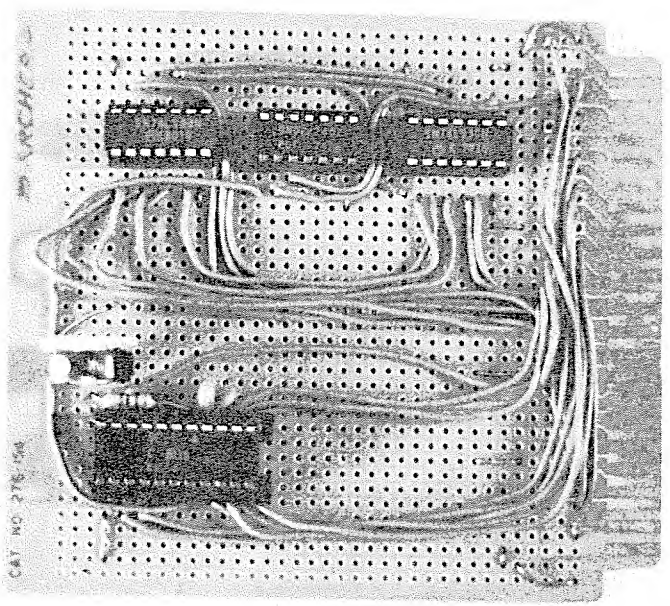


Photo 1. *The complete speech synthesizer board. At the upper left is the SC-01 speech chip, and at the right are the three address decoder chips. The trim pot at the top center adjusts overall pitch of the speech produced.*

significant two are used to provide pitch data, which in effect adds either 0, 64, 128, or 192 to the phoneme code. Thus, with one eight-bit output, a phoneme can be voiced at one of four selectable pitches.

I built my circuit on a Radio Shack plug-in PC board (see Photo 1) with enough room to add an external clock or other logic in the future. Since the address decoding is done by the 74LS variety chips, the circuit can be connected directly to the expansion port at the back of the keyboard with no additional buffering (see Photo 2).

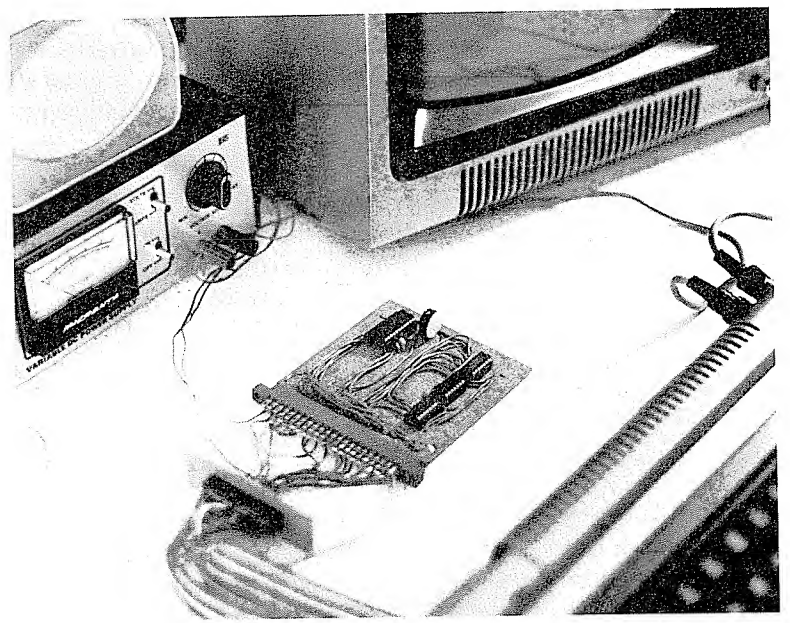


Photo 2. *The speech board as a complete system. At the upper left, the external amplifier is housed inside a speaker cabinet and beneath it is the 5-Volt power supply. The speech board is addressed by a ribbon cable from the keyboard of the TRS-80 and connects to the board with two DIP headers and two 16-pin IC sockets, allowing connection to a solderless breadboard.*

For the SC-01's audio output, I used an external amplifier to keep down the size of the overall circuit. This works fine, except that the output from the speech chip occasionally overdrives the amplifier input. If this happens, simply add a 10k pot in series with the audio output, as shown in the box marked Optional in the schematic. This measure prevents ear piercing *E* and Z sounds and makes for better sounding speech.

Software Interface

Though the Votrax data sheet doesn't help very much with programming ideas, the SC-01 is actually very easy to use. All you need to do is rewrite an English word in phonetic symbols or sounds like a dictionary pronunciation. Then the computer OUTs the specific phoneme codes in order, and the SC-01 does the rest. Most of the phoneme symbols do represent only a single written letter, but some have to be used in combination to get just one phonetic sound. Table 2 lists the combinations that produce single sounds in the order that they should be spoken.

To start talking, just convert the written message into what seem to be the correct phonemes. The more guessing and experimenting you do, the better. For example, the message *I am a speech synthesizer* becomes AH1-E1/AE-UH2-M/A-AY/S-P-E-T-CH/S-I1-N-TH-EH3-S-AH1-E1-Z-ER. The / denotes a space at this stage, and should now be converted to a *no sound* phoneme. I found that the PA0 phoneme (number 3) gives a short space good for separating words, while the PA1 (62) seems better suited for long pauses between phrases.

Sound	Phoneme Combination	Example Word
ch	T-CH	church
j	D-J	judge
f	F-H	father
a	A-AY	day, A
i	AH1-E1	find, I

Table 2. *Complex sounds*

For the SC-01 to produce these sounds, it needs to receive them in numeric form, or phoneme codes. By translating the example to code numbers and adding a PA0 space between words, the message becomes 21-60-3-46-49-12-3-32-33-3-31-37-44-42-16-3-31-11-13-57-0-31-21-60-18-58. Remember, the phonemes don't turn themselves off; so, it is necessary to send another *no sound* phoneme at the end of the message. Instead of saying *I am a speech synthesizerrrrrr*, the SC-01 voices the silence indefinitely until it gets another phoneme code. I like the PA1 (62) best because it has a longer length and can be told apart from the PA0 in a program.

Program Listing 1 gives the whole routine for the message just translated. Phonemes are read as data and then OUTed, while the asterisk serves as an end-of-data flag. The routine outputs the standard phoneme codes, not the pitch data added. While a normal monotone message like the one demonstrated sounds good, one with pitch variations has greater human quality and expression. Besides, adding the pitch changes is no great task.

Adding Emotions

By analyzing an average spoken message, you can more or less guess at where emphasis is placed and where a voice would rise or fall. Using the same example with this principle, each of the phonemes should eventually fall into one of the four available pitch levels. The last program listing used only monotone sounds and had no pitch level added, making the voice level 0.

By listening to how a person might say a phrase, you can assign each syllable a pitch level. Figure 2 shows a typical arrangement for the example I have used so far. Since the speech must rise and fall in pitch, an average level is needed. I generally put the unchanging phonemes at the + 128 level, leaving one rise and two fall positions available. This makes pitch assignments simple.

Referring to a chart such as Figure 2, just add the pitch level number to each phoneme and insert it into a DATA statement. The example message would then become the DATA lines 10–25 in Program Listing 2. Translation is now complete with not only electronic speech, but also with expression.

The basic conversion is more or less a trial and error process when the best possible speech is desired. Try different phoneme combinations to get the sound you want—and be patient! Different time durations of the same basic sound improve the flow and accent of the voice. Finally, try various pitch levels to simulate emotions by adding inflections to the machine's voice.

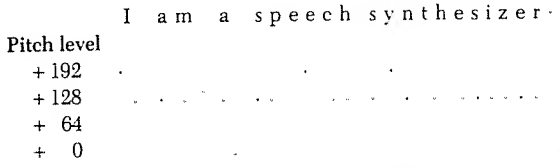


Figure 2. *Phoneme codes sent to the SC-01 equal the standard phoneme codes plus the pitch level values, as in this example of varying pitch.*

Since all of the speech produced by the SC-01 is built on basic phonemes alone, your speech need not be limited to English. Any language with an unlimited vocabulary can be voiced with the use of a pronouncing dictionary. Accent marks in foreign pronunciations should help in placing pitch changes within the speech. Though I'm no expert with foreign languages, I was able to get my 80 to speak in at least three different languages. Program Listing 3 gives the data needed for a few salutations, providing the start for a good demonstration program of the SC-01's capabilities.

You can order the SC-01 from: The Micromint Inc., 917 Midway, Woodmere, NY 11598.

Program Listing 1. *Sample message. "I am a speech synthesizer."*

```
 1 REM  SPEAK FOR YOURSELF
 2 REM  DAVID HUCABY
 3 REM  * PROGRAM LISTING 1 *
10 DATA AH1,21,E1,60,/,3,AE,46,UH2,49,M,12,/,3
15 DATA A,32,AY,33,/,3,S,31,P,37,E,44,T,42,CH,16,/,3
20 DATA S,31,I1,11,N,13,TH,57,EH3,0,S,31,AH1,21,E1,60,Z,18,ER,58,.,
   62,*
30 READ A$:
   IF A$ = "*"
    THEN
     END
40 READ A:
   OUT 3,A:
   GOTO 30
```

Program Listing 2. *Program with pitch level changes*

```
 1 REM  SPEAK FOR YOURSELF
 2 REM  DAVID HUCABY
 3 REM  * PROGRAM LISTING 2 *
10 DATA AH1,213,E1,188,/,3,AE,174,UH2,177,M,140,/,3
15 DATA A,160,AY,33,/,3,S,159,P,165,E,236,T,170,CH,144,/,3
20 DATA S,159,I1,203,N,141,TH,185,EH3,128,S,159,AH1,149,E1,188,Z,14
   6,ER,58
25 DATA .,62,*
30 READ A$:
   IF A$ = "*"
    THEN
     END
40 READ A:
   OUT 3,A:
   GOTO 30
```

Program Listing 3. *Salutations*

```
    1 REM  SPEAK FOR YOURSELF
    2 REM  DAVID HUCABY
    3 REM  * PROGRAM LISTING 3 *
    5 REM  --I AM A SPEECH SYNTHESIZER--
   10 DATA AH1,213,E1,188,/,3,AE,174,UH2,177,M,140,/,3
   15 DATA A,160,AY,33,/,3,S,159,P,165,E,236,T,170,CH,144,/,3
   20 DATA S,159,I1,203,N,141,TH,185,EH3,128,S,159,AH1,149,E1,188,Z,14
      6,ER,58
   25 DATA .,62
   30 REM  --BON JOUR...PARLEZ-VOUS FRANCAIS?--
   35 DATA B,78,UH2,113,H,91,NG,84,J,90,H,91,U1,247,R,43,.,62
   40 DATA P,101,AH2,136,R,171,L,88,A,96,AY,97,V,79,U1,183,U1,183,/,3
   45 DATA F,93,H,91,R,107,AW1,83,NG,84,S,95,A,160,AY,225,AY,225,.,62
   50 REM  --BUENOS DIAS, SENOR--
   55 DATA B,78,W,109,A,160,AY,225,N,77,O,102,S,95
   60 DATA D,222,E,236,UH1,114,S,95,/,3
00065 DATA S,95,AY,97,N,77,Y,105,O1,181,R,107,R,107,.,62
00070 REM  --HAVE A NICE DAY--
00075 DATA H,219,AE,238,V,207,A2,133,E1,188,N,77,AH1,85,E1,124,S,95,/,
      3
00080 DATA D,222,A,224,E,44,.,62,*
00100 READ A$:
      IF A$ = "*"
       THEN
        END
00110 READ A:
      OUT 3,A:
      GOTO 100
```

TUTORIAL

Computer Number Systems
And Arithmetic Operations—Part I
Down in the Dumps: Examining Memory

Computer Number Systems and Arithmetic Operations—Part I

by Gene Kovalcik

Digital computers can do little more than move numbers and add. Worst of all they handle and add only 0s and 1s, that is, binary digits or bits. Manipulating characters and English words including decimal numbers requires manipulating binary numbers.

Part I of this article covers the following subjects:

- Basic characteristics of the binary, octal, decimal, and hexadecimal number systems.
- Procedures for converting binary, octal, or hexadecimal numbers to decimal numbers.
- Procedures for converting decimal numbers to binary, octal, and hexadecimal numbers.
- How binary numbers can use octal and hexadecimal as shortcut notations.

The material in Part II includes the following:

- How the principles of decimal arithmetic operations can be applied to any number base.
- How to add and subtract binary, octal, and hexadecimal numbers.
- How to use the expanded form method to check the answers in adding and subtracting numbers in any number system.
- How to do two's complement for binary subtraction.

The decimal number system consists of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, each of which has a specific value. When two or more decimal digits are used together, the value of each digit depends upon its position as well as its digit value.

Computers use the base 2, or binary, number system. An important part of the computer machine is not arithmetical, but logical in nature. Logic, being a yes/no action, is fundamentally binary, 1s and 0s which involve electrical circuitry to be turned on and off. Figure 1 illustrates these two states.

The binary number system is used to express numeric data and in character-code schemes. Each character is a letter, symbol, punctuation mark, or number represented in binary as a zone-and-digit combination of 1s and 0s, binary digits, or bits for short. A fixed number of adjacent bits, usually eight, that represent a meaningful English character is commonly called a byte. A byte is also considered a unit of information. All computers use coding schemes to hold data and programs in computer memory and to input and output data to external devices.

Light bulb	off = 0	on = 1
Switch	off = 0	on = 1
Transistor	not conducting = 0	conducting = 1
Magnetic core	clockwise magnetism = 0	counter-clockwise magnetism = 1

Figure 1. *Relationship between electrical devices (components) is "off" and "on" to binary numbers 0 and 1.*

Number System Characteristics

A number system is a way to represent quantities of physical items. There are two characteristics of all number systems that are suggested by the value of the base. Regardless of the number system, the value of a base determines the number of different digits, or number symbols, available in the numbering system. The second characteristic is that the maximum value of a single digit is always equal to one less than the value of the base. In other words, if you count zero as the first digit or number symbol, the total number of digits must equal the value of the base.

Figure 2 shows that the base 10 number system has numbers 0 through 9, that is, 10 digits. The largest digit is 9, which is one less than the base, 10.

The general number system characteristics, also apply to binary, octal, and hexadecimal numbers. In the binary system, the digit choices are 0 and 1. The largest digit, 1, is one less than the value of the base. Any value greater than 1 must be represented by more than 1 digit, just as in decimal, any value over 9 requires more than one digit or number symbol. Refer to Figure 2 for the characteristics of base 2.

The base 8, or octal, number system is also consistent with the general number system characteristics. As indicated in Figure 2, a base of 8 shows that there are eight digit choices in the number system. The eight digits are 0 through 7. The largest single symbol in base 8 is equal to 7, or one less than the base, 8.

The hexadecimal number system, or base 16, is probably unfamiliar to you. There are 16 single-character digits or symbols. The first 10 digit choices are 0 through 9, similar to the decimal system, but an additional six digits are required for hexadecimal numbers. For convenience, the additional six symbols had to be available on computer keyboards and printers and have a commonly known sequence of single symbols. The letters A through F fulfill these requirements. See Figure 2. For hexadecimal numbers, therefore, the letters A through F are used as digits. The digit A has a decimal equivalent value of 10, and the hexadecimal F equals 15 decimal. B, C, D, and E have respective decimal values of 11, 12, 13, and

Base 10
10 digit or number symbol choices:
0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
Digit 9 is one less than 10.
10 is equal to the base.

Base 2
Two digit or number symbol choices:
0 and 1
Digit 1 is one less than 2.
2 is equal to the base.

Base 8
Eight digit or number symbol choices:
0, 1, 2, 3, 4, 5, 6, 7
Digit 7 is one less than 8.
8 is equal to the base.

Base 16
16 digit or number symbol choices:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A(10), B(11), C(12), D(13), E(14), and F(15)
Digit F (or 15 decimal) is one less than 16.
16 is equal to the base.

Figure 2. *Characteristics of decimal, binary, octal, and hexadecimal numbers.*

14. The largest single digit is number symbol F, or 15, which is one less than the value of the base, 16.

Base 8 and base 16 are number systems commonly used as shortcut notations for binary. One byte can be represented by a pair of octal numbers or by a pair of hexadecimal numbers. These shortcut notations are used because a printout of computer memory in binary numbers would be unwieldy in length.

Converting Between Number Systems

Any whole number (integer) value in one number system can be represented in any other number system. The computer's usual input and final data output values are decimal values; therefore, conversion to and from other systems is necessary. Other conditions also occur which make it convenient to convert from a decimal system to another number system. Many methods and techniques are available that can be utilized to convert from one base to another.

Converting to Base 10

To convert numbers to decimal values, it is necessary to determine column values or place values. The value of a number symbol that corresponds

to its position in a number is called its place value. The decimal number system, for example, is a place-value or positional number system in that the actual value of a specific digit is determined by:
1) The place or column the digit holds in the number.
2) The actual value of the digit.
Figure 3 shows the general rule for determining column values and the specifics for base 10, base 2, base 8, and base 16.

The general rule for determining column values is that the first (or right-hand) whole number column is equal to the value of the base to the zero power. Any number taken to the zero power is equal to 1, as is indicated in Figure 3 and the rightmost column. The second column is equal to the value of the base to the first power which is, of course, always equal to the base. The value of any number to the first power is equal to the same number.

Base ^{4}	Base ^{3}	Base ^{2}	Base ^{1}	Base ^{0}
		Base 10		
10^4	10^3	10^2	10^1	10^0
(10,000)	(1000)	(100)	(10)	(1)
		Base 2		
2^4	2^3	2^2	2^1	2^0
(16)	(8)	(4)	(2)	(1)
		Base 8		
8^4	8^3	8^2	8^1	8^0
(4096)	(512)	(64)	(8)	(1)
		Base 16		
16^4	16^3	16^2	16^1	16^0
(65,536)	(4096)	(256)	(16)	(1)

Figure 3. *Column values for bases 10, 2, 8, and 16. The equivalent decimal values of bases taken to powers (or exponents) are parenthesized for each column. Superscripted numbers show power values. For example,* $10^4 = 10 \times 10 \times 10 \times 10 = (10{,}000)$.

The values of the third and fourth columns, from right to left, continue to increase by one or more powers for each column. For example, the third column is equal to the base value times itself, the fourth-column value is the base to the third power, the fifth column is the base to the fourth power, and so on.

Let us return to decimal numbers to analyze the general rule for determining the column values, and to determine the actual meaning of the decimal number 72,083 in terms of weighting or weights concept. Imagine the string of decimal digits 7, 2, 0, 8, and 3 as representing weights according to their correspondence as column values from right to left. (See Figure

3.) The digit 7, for example, is in the fifth column and has a place value of 10,000 and an actual single digit value of 7. The 7 in the fifth place represents 7 × 10,000 or 70,000.

The second digit from the left, the 2, has a place value of 1000. The total value is 2 × 1000 or 2000. The total value of the third column is 0 × 100 or 0. The digit 8, in the second place, represents a weight of 10. The total is 8 × 10 or 80. The digit 3, in the first place, represents a weight of 1. The total is 3 × 1 or 3. Notice that the weights in the decimal number system are the powers of 10 as shown in Figure 3. Finally, the total value of any decimal number can be determined by evaluating each number in expanded format. This evaluation is applicable to any number system. The following example shows decimal 83,205 in expanded form.

$$\begin{aligned} 83{,}205 &= (8 \times 10^4) + (3 \times 10^3) + (2 \times 10^2) + (0 \times 10^1) + (5 \times 10^0) \\ &= 80{,}000 \quad 3{,}000 \quad 200 \quad 0 \quad 5 \end{aligned}$$

The binary number system consists of the digits 0 and 1 and based weights which are derived from multiplication by 2. The system is based on powers of 2 rather than 10.

Binary to Decimal Conversion

Figure 4 illustrates three steps used to convert the binary number 11011 to a decimal number. This number is not to be read as a decimal number unless the number is shown with a subscript of 10. Remember that a binary number is a sequence of binary digits, 0 and 1. Let's evaluate these digits with respect to their place or column values as we did with the decimal numbers.

$11011_2 = N_{10}$

Step 1 Determine column (decimal) value of each digit.

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
1	1	0	1	1

Step 2 Multiply column (decimal) values by digit in each of the columns.

16	8	4	2	1
×1	×1	×0	×1	×1
16	8	0	2	1

Step 3 Add the products calculated in step 2. The total is the equivalent value in decimal.

$16 + 8 + 0 + 2 + 1 = 27_{10}$

Figure 4. *A binary-to-decimal conversion in detail. Subscripts indicate the base of the number; superscripts indicate exponents.*

The first step is to determine the column values. The first column is always 1. Two times the first-column value (1) is equal to the value of the second column (2). Two times the second-column value (2) is equal to the third-column value (4). Two times the third-column value (4) is equal to the fourth-column value (8). Two times the fourth-column value (8) is equal to the fifth-column value (16). The column values double because the base is 2.

The second step in our conversion procedure is to multiply the column values by the column digits. The third step is to add the products from step 2. The total is 27, the decimal equivalent of binary 11011. You also can determine the decimal equivalent of a binary number by writing out and evaluating the number in expanded form as follows:

$$
\begin{aligned}
10110111_2 &= N_{10} \\
&= (1\times 2^7) + (0\times 2^6) + (1\times 2^5) + (1\times 2^4) + (0\times 2^3) + (1\times 2^2) + (1\times 2^1) + (1\times 2^0) \\
&= \quad 128 \qquad 0 \qquad 32 \qquad 16 \qquad 0 \qquad 4 \qquad 2 \qquad 1 \\
&= 183_{10}
\end{aligned}
$$

Octal to Decimal Conversion

Now we convert octal 257 to its equivalent decimal value using the expanded form. The solution is as follows:

$$
\begin{aligned}
257_8 &= (2\times 8^2) + (5\times 8^1) + (7\times 8^0) \\
&= (2\times 64) \quad (5\times 8) \quad (7\times 1) \\
&= \quad 128 \qquad 40 \qquad 7 \\
&= 175_{10}
\end{aligned}
$$

The first step is to determine the column values. The second step is to multiply each column value by its column digit. The final step is to add the products of step 2. The answer is the decimal number 175.

Hexadecimal to Decimal Conversion

We will now convert hexadecimal 2B3E to its equivalent decimal value. I will go through the steps in detail, before using the expanded form, because hexadecimals are unfamiliar to most readers.

The first step is to determine the column value, in decimal, of each digit. The first column of any numbering system is equal to 1. Remember that each column is worth the value of the column to its right, times the base. Here the first-column value is 1, the second-column value equals 16 (16×1), the third-column value is 256 (16×16), and the fourth-column value is 4096 (16×256). The second step is to multiply the column values determined in step 1 by the digit in the column. Note that the hexadecimal digits A through F must be converted to their equivalent decimal values of 10 through 15 before multiplying. In this case, B becomes 11 and E becomes 14. The third step is to add the products from step 2. The sum of the products in this example is decimal 11,070 which is equal to hexadecimal 2B3E.

Powers of 2	Powers of 8
$2^0 = 1$	$8^0 = 1$
$2^1 = 2$	$8^1 = 8$
$2^2 = 4$	$8^2 = 64$
$2^3 = 8$	$8^3 = 512$
$2^4 = 16$	$8^4 = 4{,}096$
$2^5 = 32$	$8^5 = 32{,}768$
$2^6 = 64$	$8^6 = 262{,}144$
$2^7 = 128$	$8^7 = 2{,}097{,}152$
$2^8 = 256$	$8^8 = 16{,}777{,}216$
$2^9 = 512$	
$2^{10} = 1{,}024$	

Powers of 10	Powers of 16
$10^0 = 1$	$16^0 = 1$
$10^1 = 10$	$16^1 = 16$
$10^2 = 100$	$16^2 = 256$
$10^3 = 1{,}000$	$16^3 = 4{,}096$
$10^4 = 10{,}000$	$16^4 = 65{,}536$
$10^5 = 100{,}000$	$16^5 = 1{,}048{,}576$
$10^6 = 1{,}000{,}000$	$16^6 = 16{,}777{,}216$

Figure 5. *Powers of 2, 8, 10, and 16 are shown. These powers or exponents of base numbers are taken to a convenient value for use in this article.*

Here is the solution using the expanded format:

$$\begin{aligned} 2B3E_{16} &= (2\times16^3) + (11\times16^2) + (3\times16^1) + (14\times16^0) \\ &= (2\times4096) \quad (11\times256) \quad (3\times16) \quad (14\times1) \\ &= \quad 8192 \quad\quad 2816 \quad\quad 48 \quad\quad 14 \\ &= 11{,}070_{10} \end{aligned}$$

Taking powers of 16 is difficult. There are tables of the powers of numbers in many mathematical books. Figure 5 illustrates powers of 2, 8, 10, and 16 to provide a convenient list for use here.

Converting from Base 10

Many times the computer technician or programmer needs to convert a decimal number to binary, octal, or hexadecimal. There is a technique for converting called the division-remainder technique. Several steps are involved in this technique.

The first step is to divide the decimal number to be converted by the value of the new base. If the conversion is to binary, divide by 2; if the conversion is to octal, divide by 8; if the new base is 16, divide by 16. For step 2, record the remainder from step 1 as the rightmost digit of the new base number. This becomes the digit in the first column from the right. Remember that

when you divide by 2, the remainder must be 0 or 1; when dividing by 8 for octal conversion, the possible remainders range from 0 through 7; when dividing by 16 for hexadecimal conversion, the possible remainders range from 0 through 15 (F). In each case, the number of possible remainders is equal to the number of digits in the new number system. The third step is to divide the answer from the previous division by the new base.

For the fourth step, record the remainder from step 3 as the next digit (to the left) of the new base number. Repeat the third and fourth steps, noting remainders from right to left, until you get an answer of 0 in the third step. Remember to write down the remainder when the division gives an answer of 0. To obtain a better understanding of this technique, let us convert decimal 36 to binary, decimal 316 to octal, and decimal 831 to hexadecimal. The three problem solutions follow:

Problem 1

$36_{10} = N_2$

```
                18
Step 1   2⌈36      Step 2   36₁₀ = 0₂
          2
          16
          16
           0 (remainder)

                9
Step 3   2⌈18      Step 4   36₁₀ = 00₂
          18
           0

                4
Step 3   2⌈9       Step 4   36₁₀ = 100₂
           8
           1

                2
Step 3   2⌈4       Step 4   36₁₀ = 0100₂
           4
           0

                1
Step 3   2⌈2       Step 4   36₁₀ = 00100₂
           2
           0

                0    Final
Step 3   2⌈1       Step 4   36₁₀ = 100100₂
           0
           1 (final remainder)
```

Note that in this example, a total of six division steps are required. You can check the answer using the expanded form to convert binary 100100 to decimal. The technique is as follows:

$$\begin{aligned} 100100_2 &= N_{10} \\ &= (1\times 2^5) + (0\times 2^4) + (0\times 2^3) + (1\times 2^2) + (0\times 2^1) + (0\times 2^0) \\ &= \quad 32 \qquad 0 \qquad 0 \qquad 4 \qquad 0 \qquad 0 \\ &= 36_{10} \end{aligned}$$

Problem 2

$316_{10} = N_8$

Step 1 $8\overline{)316}$ (39; 24; 76; 72; 4)

Step 2 $316_{10} = 4_8$

Step 3 $8\overline{)39}$ (4; 32; 7)

Step 4 $316_{10} = 74_8$

Step 3 $8\overline{)4}$ (0; 0; 4 (final remainder))

Final Step 4 $316_{10} = 474_8$

Check the answer by expanded form as follows:

$$\begin{aligned} 474_8 &= (4\times 8^2) + (7\times 8^1) + (4\times 8^0) \\ &= (4\times 64) \quad (7\times 8) \quad (4\times 1) \\ &= \quad 256 \qquad 56 \qquad 4 \\ &= 316_{10} \end{aligned}$$

Problem 3

$831_{10} = N_{16}$

Step 1 $16\overline{)831}$ (51; 80; 31; 16; 15 (F))

Step 2 $831_{10} = F_{16}$

Step 3 $16\overline{)51}$ (3; 48; 3)

Step 4 $831_{10} = 3F_{16}$

Step 3 $16\overline{)3}$ (0; 0; 3 (final remainder))

Final Step 4 $831_{10} = 33F_{16}$

Here is the check of the answer by using expanded form:

$$
\begin{aligned}
33F_{16} &= (3 \times 16^2) + (3 \times 16^1) + (15 \times 16^0) \\
&= (3 \times 256) \quad (3 \times 16) \quad (15 \times 1) \\
&= 768 \qquad 48 \qquad 15 \\
&= 831_{10}
\end{aligned}
$$

Shortcut Notations

It is very useful to have a shortcut notation for binary numbers when, for example, you want to print out the contents of computer memory or to analyze data representations. Printing the contents of memory in binary would require pages of 0s and 1s. To reduce the volume and printout time of the memory dump, and to simplify the display and analysis of data, either the octal or hexadecimal number system is utilized as a shortcut notation. Computers use either octal or hexadecimal as shortcut notation, depending upon the memory organization of the machine. If the basic unit of storage is a group of eight-bit strings, or bytes, and is designed to be a multiple of three bits, octal is used as the shortcut. If the basic unit of storage is a multiple of four bits (a dual grouping per bytes), hexadecimal is used as the shortcut

Binary	Octal	Hexadecimal	Decimal
N_1	N_8	N_{16}	N_{10}
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15
10000	20	10	16
10001	21	11	17
.	.	.	.
.	.	.	.
.	.	.	.

Table 1. *Relationships of binary, octal, hexadecimal, and decimal numbers.*

notation. Hexadecimal is more efficient. Table 1 illustrates the relationships of octal and hexadecimal to binary. For example, the maximum value of three digits of binary is equal to the maximum octal value (7). If octal digits are substituted for binary digits, the substitution is on a one-to-three basis. As a result, if octal is used as the shortcut notation, the binary contents of memory requires one third of the space and time that binary uses.

Refer to Table 1 and notice that the maximum value of one digit in hexadecimal is equal to the maximum of four bits in binary. Therefore, the value range of one digit of hexadecimal is equivalent to the value range of four binary digits. Thus, hexadecimal shortcut notation is a one-to-four reduction in space and time.

Octal Notation

In converting from binary to octal, the first step is to divide the binary digits into groups of three, starting from the right. If the number of binary digits is not a multiple of 3, zeros may be added on the left-hand side. The second step is to convert each group of three binary digits into one octal digit. Each group is treated as a separate entity. The rightmost bit of the group has a column value of 1; the second column a value of 2; the third column a value of 4. As Table 1 shows, octal numbers up through 7 are equal to decimal digits directly. The following example shows the conversion of binary 11010110 to octal:

0 1 1	0 1 0	1 1 0	← Eight-bit number grouped
2 1	4 2 1	4 2 1	← Column (decimal) values
3	2	6	← Octal number (three-digit)

A binary number that cannot be broken into three-bit groups is extended on the left side with zeros as shown by the dotted boxes. Thus, binary 11010110 equals octal 326.

To convert from octal to binary, you first change each octal digit to a three-digit binary number. Next combine the resulting groups into a single binary number. The following example illustrates how to convert octal 437 to binary.

First write each octal digit as three binary digits:

4 = 100 3 = 011 7 = 111

Run binary groups together as:

100	011	111
4	3	7

$437_8 = 100011111_2$

Hexadecimal Notation

To reduce binary to hexadecimal, first divide the bits into groups of four, and count off the bit groups, starting from the right. The second step is to

convert each group of four bits to one hexadecimal digit. The binary column values from right to left are 1, 2, 4, and 8. For convenience, use the binary-to-decimal technique explained earlier, but remember that resulting decimal values 10 to 15 should be written as hexadecimal digits A to F. To change from hexadecimal to binary, convert the decimal equivalent of each hexadecimal digit to four binary digits. The following problems illustrate the procedure.

Problem 1

Convert binary 110101110101 to hexadecimal.
Larger binary numbers are converted by separating the numbers into groups of four bits each from right to left.

1 1 0 1	0 1 1 1	0 1 0 1	← 12-bit number groupings
8 4 2 1	8 4 2 1	8 4 2 1	←Column values
D	7	5	← Hexadecimal number

Symbol D has a decimal value of 13.
$110101110101_2 = D75_{16}$

Problem 2

Convert hexadecimal 6BF to binary.
The reverse process is simply a matter of taking each hexadecimal digit and writing its equivalent four-digit binary number.

6	B	F	← Hexadecimal
0 1 1 0	1 0 1 1	1 1 1 1	← Binary

Putting the groups together produces binary 011010111111, the equivalent of hexadecimal 6BF.

Part II of this article, published in Volume 9 of the Encyclopedia, describes arithmetic operations.

Down in the Dumps: Examining Memory

by Allan S. Joffe W3KBM

Once you have used your TRS-80 for a while, the fact that it has two different types of memory comes as no surprise. The existence of BASIC in Read Only Memory (ROM) and the existence of Random Access Memory (RAM), which is memory that can be written to and read out of, becomes an accepted fact of life. The idea of the difference between volatile and non-volatile memory becomes a fact of life the first time a power interruption causes the RAM to lose its memory.

The PEEK function returns a value in a given memory location (ROM or RAM) as a decimal number ranging from 0 to 255. Since each memory cell is eight binary bits wide, the binary representation of these two extremes ranges from 00000000 to 11111111. Since the memory cell is packed with bits, the ROM must contain a translation routine so that the results of POKE appear on the screen as a decimal presentation of the binary contents of the memory cell.

Consider this programming fragment.

```
5 CLS
10 FOR X = 0 TO 15
20     J = PEEK(X)
30     PRINT J;
40 NEXT
```

If you run this program, the screen displays the following 16 values.

```
243  175  195  116  6  195  0  64  195  0  64  225  233  195
159  6
```

Note that there are 16 values because the computer sees 0 as a legitimate state or number value so that the expression FOR X = 0 TO 15 represents 16 different memory locations, specifically, the first 16 memory locations in your BASIC ROM. The ability of the computer to access the contents of the memory cells in this fashion permits you to dump the memory either to the screen or to the printer, or to both the screen and the printer. This decimal form of the memory dump has some value as a diagnostic aid for programs which have a machine-code routine embedded in a DATA statement. Such routines are generally POKEd into a section of memory. If you want to check on the accuracy of the POKE procedure, the decimal form of the memory dump might be useful.

A far more practical form of the memory dump presents the dump in hexadecimal or base 16, form. Let us examine the same 16 ROM locations and see what they look like if you dump them in the hex format.

```
F3  AF  C3  74  06  C3  00  40

C3  00  40  E1  E9  C3  9F  06
```

The information in the dumps is identical. Only the form of the presentations differs. The value lies in the fact that the hex format is the one you will encounter most often in computer discussions, particularly when machine code or assembler code is the topic. Converting the hex value in any location to its equivalent decimal value is no great trick.

Both systems have positional weighting. For example, the decimal value of 243 really says that the rightmost position has a value of 3 times 1. The next position has a value of 4 times 10, and the last position has a value of 2 times 100. The values 1, 10, and 100 are the weighting values of the position, and the absolute value of any digit in any position is the product of that digit times the weighting value for the position. The total value of the sum of these products is the value of the number as a whole.

The hex system weighting value for the rightmost digit is 1, the same as the decimal system. The next digit weighting value, however, is 16 since hex is a base 16 number system. Since an eight-bit memory cell cannot hold a number greater than 255, and since 255 can be represented by two hex digits, we can stop at the second digit for now.

Since we need 16 characters to represent the 16 states of value in base 16, we use the numerals 0 through 9 to represent the first 10 states and the letters A through F for values from 10 to 15.

Note that our two listings show that 243 decimal is equal to F3 hex. We can convert from hex to decimal using our knowledge of the hex weighting positional values. The rightmost digit has a total value of 3 times 1. The second digit, F, has a value of 15 times its positional weight of 16, which equals 240. Add the value of 3 from the first digit, and you have converted the hex value F3 to its decimal equivalent of 243.

Since PEEK does a good job of giving us a memory cell value in decimal, all we need to turn the PEEK value into a hex value is an algorithm that the computer can handle easily. An easy way to see the development of a possible algorithm is to explore how we might take a hex number from 0 to 255 decimal or from 00 to FF hex and convert the decimal value delivered by the PEEK function into hex.

Staying with the value of 243 decimal, we must find out how many times it is divisible by 16. This gives us the most significant bit (MSB) of the number or that portion of the number which holds the greatest portion of the total value of the entire number. For example, in the decimal value 92, the 9 is the MSB and represents a value of 90, while the 2, which is the least

significant bit (LSB), represents only a value of 2. If we perform the suggested division on the value of 243 (243/16), our answer is 15.1875. We take the integer value of this division which is 15, and this gives us the value of the MSB of the hex value. Since the value of 15 decimal represents itself in hex as the letter F, the MSB is F. The remainder of the division contains the information needed to construct the LSB to finish the conversion. When we multiply the decimal value .1875 by 16, the result is 3; thus the hex equivalent of 243 decimal is F3.

Remember that our memory cell contains eight bits. In hex terms, the values range from 0 to F for any single digit. In binary or bit terms, this range of values can be represented by combinations of four bits, ranging from 0000 to 1111; thus you can see that any eight-bit memory cell can hold two of these four-bit digit representations whose hex representations range from 00 to FF or from 0 to 255 decimal. Once again, this range is actually 256 different possible combinations since 0 is very real to a computer.

Consider this initial algorithm to convert the decimal value produced by PEEK to its equivalent hex value.

```
5 CLS
10 INPUT J
20 D = PEEK(J):' THIS GETS A VALUE FROM THE MEMORY
25 '                CELL SPECIFIED BY THE VALUE OF J
30 MSB = INT(D/16)
40 LSB = D-16*MSB
50 PRINT MSB;LSB
60 GOTO 10
```

Run this and try to examine the first 16 locations in ROM by entering successive values for J from 0 to 15. The following results.

```
15  3   10  15   12  3   7  4   0  6   12  3   0  0   4  0
12  3   0   0    4   0   14 1   14 9   12  3   9  15  0  6
```

The computer has followed its instructions faithfully, but it is obvious that something has to be added. It works as long as a particular value is less than 10, but for values in excess of 9, we must program the computer to display such values in terms of hex notation. This means that a value from 10 to 15 must be displayed as a letter in the range from A to F.

The computer recognizes certain ASCII values for the numerals 0 through 9 and the letters A through F. There are other letters and symbols that have ASCII values, but the 16 symbols that make up the hex number set are those of immediate interest. The ASCII values for 0 through 9 are values from 48 through 57. The corresponding values for the letters A through F are from 65 to 70. In the command mode, if we type PRINT CHR$(65) and press ENTER, we see an A on the screen. If we had used the expression CHR$(48), we would have seen a 0 printed on the screen. In order to get the proper representation of the hex values we need to make our first algorithm work, we have to add 48 to any value produced if that value is to represent a

numeral from 0 to 9. Thus we produce the ASCII range of values from 48, for 0, to 57, for 9. If the algorithm produces values from 10 to 15, we have to add 55 to these values to produce the ASCII values of 65, for A, to 70, for F.

With the ASCII values and the CHR$ function of the computer, along with fundamental relational operators, we can produce an algorithm that gives us a printout of the desired memory dump in hex notation.

```
5 CLS
10 FOR J=0 TO 15
20 D= PEEK(J)
30 MSB= INT(D/16)
35 LSB=D-16*MSB
40 IF MSB<10 THEN MSB=MSB+48:GOTO 60
50 IF MSB>9 THEN MSB=MSB+55
60 PRINT CHR$(MSB);
80 IF LSB<10 THEN LSB=LSB+48:GOTO 100
90 IF LSB>9 THEN LSB=LSB+55
100 PRINT CHR$(LSB);
110 PRINT CHR$(32);
120 NEXT J
130 END
```

Notice that we are again looking at the contents of the first 16 ROM locations. We have included the generation of the ASCII values needed to work in relation with the CHR$ function so that the computer can tell when a developed value is a numeral from 0 to 9 or a value that needs translation (10 to 15) into a letter (A to F) for proper presentation of a hex formatted dump. We also have our relational expressions in lines 40 and 50 for the MSB and in lines 80 and 90 for the LSB. Line 110 contains the ASCII value for a space so that after the LSB and MSB have been printed on the screen, we then have a space before the next information appears on the screen. This raises an important issue in any program that offers a display, namely that of screen formatting. Raise the number of ROM memory cells to be examined by changing line 10 to read:

```
10 FOR J = 0 TO 31
```

This would have us examine a total of 32 ROM cells. The display which results is anything but an object of either beauty or utility. If you run the program with this revision, you see an entire line of print on the screen which ends with a split pair hex number, and then the balance of the values are printed on the next line. It is obvious that a fix is in order. It is to have 16 hex numerals per line. You must add a counter that tells the computer that 16 pairs of hex digits have been printed. At that point in the program, the computer generates a carriage return and line feed so that the next 16 or so digits are printed on the screen line. You can do this by adding:

```
115 C = C + 1:IF C = 16 THEN C = 0:PRINT CHR$(10);
```

Note that if you omit the final semicolon of this line, the lines of print on the screen are double-spaced, but with the semicolon they are single-spaced. If

you leave the semicolon in line 115 and beef up line 10 to examine the first 224 ROM memory locations, you get a total of 14 lines of digits on the screen. The new version of line 10 is:

```
10 FOR J=0 TO 223
```

As the program is, it takes about 18 seconds to hex dump the 224 memory locations to the screen.

Adding a line:

```
7 DEFINT J,D,C
```

shaves about one second from this time because the computer can handle integers faster than it can handle single-precision values. By examining what seem to be minor differences, you can gain insight into increasing the speed of your BASIC routines.

If we alter the input portion of the program slightly, we can examine contiguous sections of memory at will rather than having to enter new values each time we wish to examine another contiguous section of memory. Start with a line like this:

```
INPUT"STARTING ADDRESS IN DECIMAL";J
```

Next we decide how many memory locations we wish to have printed on the screen each time the program runs. Assume that this figure is 224, which gives us 14 lines of printout. The line to do this would read as follows:

```
FOR J=J TO J+223
```

At the end of the first printout, enter a line which in its simplest form would look like this:

```
INPUT "FOR ANOTHER SECTION OF MEMORY PRESS ENTER";Z:CLS:GOTO(line number)
```

The line number in parentheses would be the line number that reads

```
FOR J=J TO J+223
```

You have a program that is easier to use if you intend to examine a large area of contiguous memory blocks. You can also use the INKEY$ function as follows:

```
130 A$ = INKEY$
140 IF A$ = "Y" GOTO 170 ELSE IF A$ = "N" THEN END ELSE GOTO 130
170 CLS:GOTO 10
```

I mention the alternative as there seems to be a large body of opinion that the INKEY$ function is classier than the INPUT function to do the job.

The following listing incorporates the main points made so far. The only new item is the INKEY$ routine, which tells you on the screen what letter to enter to make the choice of continuing the listing dump or to terminate the effort.

```
3 REM PROGRAM ILLUSTRATING IDEAS TO THIS POINT
5 CLS
7 INPUT "START ADDRESS IN DECIMAL";J
```

Program continued

```
10 FOR J=J TO J+223
20 D= PEEK(J)
30 MSB= INT(D/16)
35 LSB=D-16*MSB
40 IF MSB<10 THEN MSB=MSB+48:GOTO 60
50 IF MSB>9 THEN MSB=MSB+55
60 PRINT CHR$(MSB);
80 IF LSB<10 THEN LSB=LSB+48:GOTO 100
90 IF LSB>9 THEN LSB=LSB+55
100 PRINT CHR$(LSB);
110 PRINT CHR$(32);
115 C=C+1:IF C=16 THEN C=0:PRINT CHR$(10);
120 NEXT J
130 A$=INKEY$
135 PRINT @960,"ANOTHER DUMP (Y) OR (N)";
140 IF A$="Y" GOTO 170 ELSE IF A$="N" THEN END ELSE GOTO 130
170 CLS:GOTO 10
```

The next listing contains many of the items discussed so far but differs in its programming approach in one significant regard. The author, Wayne Davis, chose to park the developing values of the hex dump in an array. The program works very well and its only problem is speed. The array is used for extra overhead for the computer. If you run this program and hex dump 224 locations, you find that the computer takes about 32 seconds to finish the dump. This compares a bit unfavorably with a previous example where the dump of 224 memory locations took about 18 seconds. For short dumps of about 32 locations (two lines of screen), the time variance between the two programs is not significant. It is only when you compare processing time for a large number of locations to be dumped that the time efficiency of the two programs makes the array approach the less desirable of the two.

```
400 CLS
450 DEFINT S,E,J,N,Q,D
500 INPUT"DECIMAL START ADDRESS";S
510 INPUT"DECIMAL END ADDRESS";E
520 FOR J= S TO E
530 D= PEEK(J)
1020 FOR N = 1 TO 2:X=INT(D/16):Q=D-16*X
1030 IF Q < 10 THEN Q = Q + 48:GOTO 1050
1040 IF Q > 9 THEN Q = Q + 55
1050 A(N)=Q:D=X:NEXT N
1060 FOR N = 2 TO 1 STEP -1:PRINT CHR$(A(N));:NEXT N
1062 PRINT CHR$(32);
1065 IF W = 16 THEN PRINT CHR$(10);:W = 0
1070 NEXT J
```

The symbols that represent 0 through 9, unlike the letter values, are already in a form that the computer can print without the apparent need for further processing. It is apparent that the program returns values for the letters A through F that need converting so that the computer prints these values as letters. The numbers 0 through 9 do not seem to need this extra step, but there is a reason for converting them.

```
5 CLS
10 FOR J = 0 TO 15
15 A = PEEK(J)
20 B = A/16
30 C = FIX(B)
```

```
35 G = A - 16 * C
40 IF C < 10 PRINT C;
45 IF C > 9 PRINT CHR$(C + 55);
50 IF G < 10 PRINT G;
60 IF G > 9 PRINT CHR$(G + 55);
65 PRINT CHR$(32);
70 NEXT J
```

This program dumps the first 16 ROM cells. You see that the printout is peculiar. The spacing is fine when a hex digit is made up of two letters but is messed up when the hex digit is other than two letters. This is a result of mixing two different methods by which the TRS-80 handles its printout forms. You can dance around the problem, but I recommend the straight approach of uniform printout style in the program.

We are still missing at least two elements that you may want in this kind of utility program. They are a printer routine and an indicator at the beginning or end of each line of just which memory location, either the first hex digit or the last hex digit of the line, is being shown. As for the memory location problem, we have two choices in that the form of the digit that identifies either the first or last cell dump on the line may be either a decimal or a hex number. Decimal is simpler, so let us tackle that one first.

```
5 CLS
6 INPUT "START ADDRESS IN DECIMAL";S
8 INPUT "END ADDRESS IN DECIMAL";E
10 FOR X = S TO E
12 IF J= 16 PRINT STRING$(4,32);:PRINT X-1;:J=0
13 IF J=0 AND X>S PRINT CHR$(10);
15 A= PEEK(X)
20 G= A/16
30 B= G-INT(G)
40 IF G-B>9  PRINT CHR$((G-B)+55);ELSE  PRINT CHR$((G-B)+48);
50 IF 16 *B>9  PRINT CHR$((16*B)+55);ELSE  PRINT CHR$((16*B)+48);
55 PRINT CHR$(32);
60 J=J+1:NEXT X
```

The program develops as before except that I have used INT instead of FIX to develop the LSB value. The elements I added to print out the location of the last hex digit on each line are the counter, J, in line 60 and the elements in line 12.

```
F3 AF C3 74 06 C3 00 40 C3 00 40 E1 E9 C3 9F 06      15
C3 03 40 C5 06 01 18 2E C3 06 40 C5 06 02 18 26      31
C3 09 40 C5 06 04 18 1E C3 0C 40 11 15 40 18 E3      47
C3 0F 40 11 1D 40 18 E3 C3 12 40 11 25 40 18 DB      63
C3 D9 05 C9 00 00 C3 C2 03 CD 2B 00 B7 C0 18 F9      79
0D 0D 1F 1F 01 01 5B 1B 0A 00 08 18 09 19 20 20      95
0B 78 B1 20 FB C9 31 00 06 3A EC 37 3C FE 02 D2      111
```

Figure 1. *Printout of first 112 ROM locations*

The sample printout (in Figure 1) shows the contents of the first 112 ROM locations with the end-of-line location identifier number. Remember that the first location is 0 which helps explain that 111 is really location 112.

When counter J increments to 16, four blanks are printed with the aid of STRING$, and then the end-of-line number is printed. Counter J is set equal to 0, and the next line of hex digits is ready to be printed out. As in a previous example, if you omit the semicolon in line 12 that appears after X-1, your lines of hex numbers will be double-spaced. If you want your end-of-line identifier in hex notation, our next stop is to examine this problem.

We need to consider printing four hex digits to get numbers large enough to act as the end-of-line identifier. We know that two hex digits can represent 255 decimal but we need four hex digits to represent a number as large as 65535. In hex, we will go from 0000 to FFFF. We know that the hex positional weights of the digits discussed so far are 1 and 16. The next two positional weights are 256 and 4096.

The maximum position values from left to right of the four digit hex number FFF are as follows:

4096 times 15 = 61440
256 times 15 = 3840
16 times 15 = 240
1 times 15 = 15

This gives us our grand total of 65535, which gives us hex values large enough to serve as memory location markers for the hex dump when we get into high-memory locations.

The following program gives end-of-line location values in hex notation.

```
5 CLS
7 INPUT "START ADDRESS IN DECIMAL";J
10 FOR J=J TO J+223
20 D= PEEK(J)
30 MSB= INT(D/16)
35 LSB=D-16*MSB
40 IF MSB<10 THEN MSB=MSB+48:GOTO 60
50 IF MSB>9 THEN MSB=MSB+55
60 PRINT CHR$(MSB);
80 IF LSB<10 THEN LSB=LSB+48:GOTO 100
90 IF LSB>9 THEN LSB=LSB+55
100 PRINT CHR$(LSB);
110 PRINT CHR$(32);
115 C=C+1:IF C=16 GOSUB 490
120 NEXT J
130 A$=INKEY$
135 PRINT @960,"ANOTHER DUMP (Y) OR (N)";
140 IF A$="Y" GOTO 170 ELSE IF A$="N" THEN END ELSE GOTO 130
170 CLS:GOTO 10
180 END
490 PRINT STRING$(4,32);:R=J
500 FOR N= 1 TO 4
510 X=INT(R/16):Q=R-16*X
520 IF Q<10 THEN Q=Q+48:GOTO 540
530 IF Q>9 THEN Q=Q+55
540 A(N)=Q:R=X:NEXT N
550 FOR N= 4 TO 1 STEP -1
560 PRINT CHR$(A(N));
570 NEXT N
575 C=0
580 PRINT CHR$(10);
590 RETURN
```

The bulk of the program should look familiar since it was demonstrated earlier. The routine that does the hex conversion for the end-of-line location marker is contained in the subroutine which starts at line 490. The STRING$ expression, as in a prior example, spaces the end-of-line marker away from the last printed memory cell dump on any given line. The conversion routine is the one I used earlier and is the creation of Wayne Davis. When I first used this routine, I pointed out that it was a bit slow compared to another routine which was demonstrated to be considerably faster. The first time I employed the routine, I used it some 224 times to dump out the contents of 224 memory locations. In this use, the routine is used only once per line, and the overhead it adds to the program's running time is well worth its simplicity. The point is that any routine has value if applied in the right spot. Do not throw away any routine you come across, for sooner or later the right spot will come up.

The following program is basically the same as the last one. Only the hex conversion routine has been changed in the subroutine. It shows that the positional weighting of the hex system of notation has been followed closely, and why I would rather type in Wayne Davis' version than this one. The relative speeds are so close that the extra typing effort is pointless.

```
5 CLS
7 INPUT "START ADDRESS IN DECIMAL";J
10 FOR J=J TO J+223
20 D= PEEK(J)
30 MSB= INT(D/16)
35 LSB=D-16*MSB
40 IF MSB<10 THEN MSB=MSB+48:GOTO 60
50 IF MSB>9 THEN MSB=MSB+55
60 PRINT CHR$(MSB);
80 IF LSB<10 THEN LSB=LSB+48:GOTO 100
90 IF LSB>9 THEN LSB=LSB+55
100 PRINT CHR$(LSB);
110 PRINT CHR$(32);
115 C=C+1:IF C=16 GOSUB 1000
120 NEXT J
130 A$=INKEY$
135 PRINT @960,"ANOTHER DUMP (Y) OR (N)";
140 IF A$="Y" GOTO 170 ELSE IF A$="N" THEN END ELSE GOTO 130
170 CLS:GOTO 10
180 END
1000 PRINT STRING$(4,32);: R=J
1020 H4=INT(R/4096)
1030 H3=INT((R-H4*4096)/256)
1040 H2=INT((R-((H4*4096)+(H3*256)))/16)
1050 H1=R-((H4*4096)+(H3*256)+(H2*16))
1060 IF H4<10 PRINT CHR$(H4+48); ELSE PRINT CHR$(H4+55);
1070 IF H3<10 PRINT CHR$(H3+48); ELSE PRINT CHR$(H3+55);
1080 IF H2<10 PRINT CHR$(H2+48); ELSE PRINT CHR$(H2+55);
1090 IF H1<10 PRINT CHR$(H1+48); ELSE PRINT CHR$(H1+55);
1100 C=0
1110 PRINT CHR$(10);
1120 RETURN
```

The last item we will tackle is putting a printer routine into the program. The following listing does the job.

```
3 REM LINE PRINTER ROUTINE ADDED
5 CLS
```

Program continued

```
7 INPUT "START ADDRESS IN DECIMAL";J:CLS
10 FOR J=J TO J+223:W=W+4
20 D= PEEK(J)
30 MSB= INT(D/16)
35 LSB=D-16*MSB
40 IF MSB<10 THEN MSB=MSB+48:GOTO 60
50 IF MSB>9 THEN MSB=MSB+55
60 PRINT CHR$(MSB);
80 IF LSB<10 THEN LSB=LSB+48:GOTO 100
90 IF LSB>9 THEN LSB=LSB+55
100 PRINT CHR$(LSB);
110 PRINT CHR$(32);
115 C=C+1:IF C=16 GOSUB 1000
120 NEXT J
125 IF PEEK(14312)<>63 THEN GOTO 130 ELSE GOSUB 2000
130 A$=INKEY$
135 PRINT @960,"ANOTHER DUMP (Y) OR (N)";
140 IF A$="Y" GOTO 170 ELSE IF A$="N" THEN END ELSE GOTO 130
170 W=0:CLS:GOTO 10
180 END
1000 PRINT STRING$(4,32);: R=J
1020 H4=INT(R/4096)
1030 H3=INT((R-H4*4096)/256)
1040 H2=INT((R-((H4*4096)+(H3*256)))/16)
1050 H1=R-((H4*4096)+(H3*256)+(H2*16))
1060 IF H4<10 PRINT CHR$(H4+48); ELSE PRINT CHR$(H4+55);
1070 IF H3<10 PRINT CHR$(H3+48); ELSE PRINT CHR$(H3+55);
1080 IF H2<10 PRINT CHR$(H2+48); ELSE PRINT CHR$(H2+55);
1090 IF H1<10 PRINT CHR$(H1+48); ELSE PRINT CHR$(H1+55);
1100 C=0
1110 PRINT CHR$(10);
1120 RETURN
2000 LPRINT CHR$(27)CHR$(66)
2010 FOR L= 15360 TO 15361+W
2020 R=PEEK(L)
2030 LPRINT CHR$(R);
2040 NEXT L
2050 FOR E= 1 TO 6:LPRINT CHR$(32):NEXT E
2055 W=0
2060 RETURN
```

By now most of the program is familiar to you. We have added the subroutine starting at line 2000. This line is a specific instruction for my printer which is an Okidata Microline 80. It sets this particular printer to print 64 characters per line. You may have to change this line to suit your printer. The following program excerpt shows the necessary revisions.

```
1990 :
     ' ******************************************************
1991 :
     ' *  CHANGES TO USE WITH OTHER THAN OKIDATA PRINTERS  *
1992 :
     ' *                                                    *
1993 :
     ' *  REPLACE STATEMENT 2000 AND INSERT 2032-2033.      *
1994 :
     ' ******************************************************
2000 CC = 0
2010 FOR L = 15360 TO 15361 + W
2020  R = PEEK(L)
2030  LPRINT CHR$(R);
2032  CC = CC + 1
2033  IF CC > 56
       THEN
        CC = 0:
        L = L + 7:
```

```
        LPRINT " "
2040  NEXT L
2050 FOR E = 1 TO 6:
      LPRINT CHR$(32):
      NEXT E
2055 W = 0
2060 RETURN
```

The balance of the subroutine is a SCREEN PRINT routine that takes the contents of the video memory area (15360 to 16383) that contains screen information and, by means of the PEEK statement in line 2020 and the next line, sends the screen information to the printer. Line 2050 is also a specific instruction for my printer which advances the paper six lines after the screen print dump to the printer is finished. Thus if you print two consecutive dumps, they will be nicely spaced on the paper, one from the other.

The expression W = W + 4 in line 10 serves as a character counter to tell line 2010 in the printer subroutine how much of the video memory to print. Notice that lines 170 and 2055 both contain the expression W = 0, which in each case resets the line character counter after each screen print dump.

Line 125 contains an important part of the logic of the program. On the Model I, if memory location 14312 is not equal to 63, the printer is not ready, off line, missing totally, or in some sort of trouble, perhaps not plugged into the AC line.

If this memory location is equal to 63, the program transfers control to the printer, and you get a printout of what is on the screen. You do not lose the contents of the screen while the printer is working. The nice part about this manner of programming in the printer routine is that if you want the printer, you put it on-line, but if you want just a video display of the dump, take the printer off-line. The program then bypasses the printer routine and proceeds as though there was no printer routine as part of the program.

I suggest that you do a bit of homework. Put your printer into various states of operation, such as no AC, AC on but off-line, and so on. Print PEEK the memory location 14312 to see if your particular printer puts the same value into this location as mine does for various situations. Revisions must also be made to the following program to account for different printers.

```
100 REM START ADDRESS IN HEX ADDED
120 CLS
140 INPUT "START ADDRESS IN hEX";H$:CLS:GOSUB 930
160 FOR J=J TO J+223:W=W+4
180 D= PEEK(J)
200 MSB= INT(D/16)
220 LSB=D-16*MSB
240 IF MSB<10 THEN MSB=MSB+48:GOTO 280
260 IF MSB>9 THEN MSB=MSB+55
280 PRINT CHR$(MSB);
300 IF LSB<10 THEN LSB=LSB+48:GOTO 340
320 IF LSB>9 THEN LSB=LSB+55
340 PRINT CHR$(LSB);
360 PRINT CHR$(32);
380 C=C+1:IF C=16 GOSUB 540
400 NEXT J
```

Program continued

```
420 IF PEEK(14312)<>63 THEN GOTO 440ELSE GOSUB 780
440 A$=INKEY$
460 PRINT @960,"ANOTHER DUMP (Y) OR (N)";
480 IF A$="Y" GOTO 500ELSE IF A$="N" THEN END ELSE GOTO 440
500 W=0:CLS:GOTO 160
520 END
540 PRINT STRING$(4,32);: R=J
560 H4=INT(R/4096)
580 H3=INT((R-H4*4096)/256)
600 H2=INT((R-((H4*4096)+(H3*256)))/16)
620 H1=R-((H4*4096)+(H3*256)+(H2*16))
640 IF H4<10 PRINT CHR$(H4+48); ELSE PRINT CHR$(H4+55);
660 IF H3<10 PRINT CHR$(H3+48); ELSE PRINT CHR$(H3+55);
680 IF H2<10 PRINT CHR$(H2+48); ELSE PRINT CHR$(H2+55);
700 IF H1<10 PRINT CHR$(H1+48); ELSE PRINT CHR$(H1+55);
720 C=0
740 PRINT CHR$(10);
760 RETURN
780 LPRINT CHR$(27)CHR$(66)
800 FOR L= 15360 TO 15361+W
820 R=PEEK(L)
840 LPRINT CHR$(R);
860 NEXT L
880 FOR E= 1 TO 6:LPRINT CHR$(32):NEXT E
900 W=0
920 RETURN
930 H= 65536:J=0
935 K= LEN(H$)
940 Y=ABS(K-4)
950 FOR G= 0 TO Y:H=H/16:NEXT G
960 FOR N= 1 TO K
970 Q=ASC(MID$(H$,N,1))
980 IF Q< 58 THEN Q=Q-48:GOTO 1000
990 Q=Q-55
1000 J= J+(Q*H):H=H/16:NEXT N
1010 RETURN
```

It is also possible to enter the starting point of the hex dump in hex. This gives me a chance to introduce yet another conversion routine. Once again, this is the work of Wayne Davis. I renumbered the fundamental program to produce a neater listing. There are two changes. In line 140, we see the invitation to enter the starting location in hex. The program then branches to a new subroutine starting at line 930. This subroutine takes the hex input value and returns an equivalent decimal value in terms of J. The balance of the program is identical to the previous listing. Figure 2 shows a sample hex dump.

```
F3 AF C3 74 06 C3 00 40 C3 00 40 E1 E9 C3 9F 06     000F
C3 03 40 C5 06 01 18 2E C3 06 40 C5 06 02 18 26     001F
C3 09 40 C5 06 04 18 1E C3 0C 40 11 15 40 18 E3     002F
C3 0F 40 11 1D 40 18 E3 C3 12 40 11 25 40 18 DB     003F
C3 D9 05 C9 00 00 C3 C2 03 CD 2B 00 B7 C0 18 F9     004F
0D 0D 1F 1F 01 01 5B 1B 0A 00 08 18 09 19 20 20     005F
0B 78 B1 20 FB C9 31 00 06 3A EC 37 3C FE 02 D2     006F
00 00 C3 CC 06 11 80 40 21 F7 18 01 27 00 ED B0     007F
21 E5 41 36 3A 23 70 23 36 2C 23 22 A7 40 11 2D     008F
01 06 1C 21 52 41 36 C3 23 73 23 72 23 10 F7 06     009F
15 36 C9 23 23 23 10 F9 21 E8 42 70 31 F8 41 CD     00AF
8F 1B CD C9 01 21 05 01 CD A7 28 CD B3 1B 38 F5     00BF
D7 B7 20 12 21 4C 43 23 7C B5 28 1B 7E 47 2F 77     00CF
BE 70 28 F3 18 11 CD 5A 1E B7 C2 97 19 EB 2B 3E     00DF
```

Figure 2. *Dump in hexadecimal format*

UTILITY

New Disk Owner's Delight
Generate
Professional Looking Listings with a Teletype™
More Patches to EDTASM

New Disk Owners' Delight

by Gerald DeConto

Many of you have finally gone on to disk systems and found what a pleasure they can be to use. You also have probably discovered that many of your favorite utilities and games are very difficult to put on disks. In fact, many are incompatible since they need to load where the DOS is located.

I was in this position a few months back and considered buying NEWDOS for its LMOFFSET/CMD which was supposed to solve this problem. Then, I realized that I could write my own tape-to-disk transfer routine. With a little experimentation and my copy of Supermap by Fuller Software, I did it. The result is T2D.

About the Program

T2D (see Program Listing) is a two-part utility. The first part, Loader, loads the SYSTEM tape program into memory so that the DOS is not wiped out. It then saves the program along with a relocator to disk as a CMD file. Since the program is stored high in memory, execution is automatic once you load it. DOS is then unnecessary; so the program is moved back to its intended location.

The second part of T2D, Mover, moves the program back to its intended location in low RAM. This relocator also fixes the BASIC vectors so that any program that needs to operate in a Level II BASIC environment is not frustrated.

Once T2D is executed, control is sent to Loader which displays the instructions and waits for you to press the ENTER key. Loader then turns on the tape and reads in the preliminaries, sending the program name to the screen. It then loads in all the program blocks and stores them in high memory exactly as they are read. See Figure 1 for the SYSTEM tape format.

The checksum bytes and block identifiers are discarded after use. The results of loading in the program are shown in Figure 2. Once the last program byte has been stored, Loader finds the end-of-program byte which signals Loader to load in the two-byte program entry address and store it at the end of Mover so that Mover's last instruction causes a jump to the start of your program.

```
:256 BYTES OF 0
:BYTE OF VALUE A5H FOR SYNCHRONIZATION
:BYTE OF VALUE 55H AS PROGRAM NAME HEADER
:SIX-BYTE (CHARACTER) NAME
:BYTE OF VALUE 3CH AS BLOCK HEADER
:BYTE CONTAINING BLOCKS LENGTH
:TWO BYTES SPECIFYING WHERE BLOCK SHOULD GO
:BLOCK OF LENGTH SPECIFIED
:CHECKSUM BYTE
:BYTE OF VALUE 3CH AS BLOCK HEADER
:BYTE CONTAINING BLOCKS LENGTH
:
:       .       (MORE OF SAME)
:       .
:BYTE OF VALUE 78H DENOTING END OF PROGRAM
:TWO-BYTE PROGRAM ENTRY ADDRESS
```

Figure 1. *SYSTEM tape format*

Storing to Disk

Once the program is set up in memory, you have to get it on disk, which is a bit harder than it sounds. You can let the computer do all the work. After looking at Supermap for a while and experimenting, I learned a few things. When a command is typed into the computer in DOS, a copy of the com-

```
7000H -LOADER IS LOCATED HERE
.
.
.
8000H -MOVER LOCATED HERE
802AH-MOVER ENDS HERE
802BH-BLOCK LENGTH          (YOUR PROGRAM BEGINS HERE)
      -TWO-BYTE DESTINATION ADDRESS OF BLOCK
      -BLOCK OF ABOVE LENGTH
      -BLOCK LENGTH
      -TWO-BYTE DESTINATION ADDRESS OF BLOCK
      -BLOCK OF ABOVE LENGTH
      -BLOCK LENGTH
          .
          .       (MORE OF SAME)
          .
      -BLOCK LENGTH OF 0
      -BLOCK DESTINATION ADDRESS OF 0000H
```

Figure 2. *Results of T2D*

mand is sent to 4318H. DOS uses this command copy for working. I also found that loading HL with a DOS command address and a jump to 4405H executes the command. To put the program-Mover combination on disk, I simply had to get Loader to create a DUMP command with the appropriate data.

Loader starts off with 4318H as the command storage address. All command parts are sent there. DUMP is the first thing stored. The program then asks for the filespec that the program is to be saved under. A filespec is necessary since it is sent to the location following DUMP. (START-X'8000', END-X) is stored just after the filespec. The command needs the last address at which data is stored. The program creates and stores an ASCII version of the address and sends the rest of the DUMP command. The result is something like this:

DUMP filespec/ext:D (START-X'8000',END-X'addr',TRA-X'8000')

HL is loaded with 4318H, and Loader jumps to 4405H to execute the DUMP command.

When the new disk version of the program is called and loaded in, several things happen. Since the program is high in memory, the DOS system is not wiped out which would cause a reboot. The Level II BASIC vectors are initialized to avoid trouble with any programs that need those vectors, and the program moves from its location high in memory to its intended location. If you design the program to load in over 7FFFH, you will have problems, but you can remedy them by putting Loader and Mover in higher. With the program in its proper location, control passes to it, and T2D is finished.

Remember that any program designed to load in over 7FFFH may not work. Also note that programs that have changing baud rates or weird storage structures such as MICROCHESS 1.5 will not load.

Program Listing. *T2D*

```
                        00100 ;***********************************************
                        00110 ;*       TAPE TO DISK CONVERTER UTILITY        *
                        00120 ;*                                             *
                        00130 ;*          BY:  G.DECONTO                     *
                        00140 ;*               2014 QUILCHENA CRES.          *
                        00150 ;*               VANCOUVER,B.C.                *
                        00160 ;*               CANADA, V6M 1E3               *
                        00170 ;*---------------------------------------------*
                        00180 ;*  REQUIRES: DISK SYSTEM,EDITOR/ASSEMBLER     *
                        00190 ;*  THIS UTILITY:1/ LOADS IN A SYSTEM PROGRAM   *
                        00200 ;*               :2/ MODIFIES IT FOR DISK      *
                        00210 ;*               :3/ SAVES THE PROGRAM AS A FILE *
                        00220 ;*               :4/ DOES ALL CHECKS           *
                        00230 ;***********************************************
7000                    00250           ORG     7000H
7000 F3                 00260 LOADER    DI                      ;STOP CLOCK
7001 CDF801             00270           CALL    1F8H            ;STOP TAPE
7004 CDC901             00280           CALL    1C9H            ;CLS
7007 21F270             00290           LD      HL,TITLE
700A CDCF44             00300           CALL    44CFH           ;PRINT TITLE
700D CDE470             00310 INPUT1    CALL    WAIT            ;AWAIT <ENTER>
7010 214B71             00320           LD      HL,CMD1         ;"DUMP "
7013 111843             00330           LD      DE,4318H        ;DESTINATION
7016 010500             00340           LD      BC,5            ;# CHARS.
7019 EDB0               00350           LDIR                    ;MOVE CHARS.
                        00360 ;---------------------------------------------
                        00370 ;       GET SYSTEM FILE FROM TAPE
                        00380 ;---------------------------------------------
701B AF                 00390           XOR     A
701C CD1202             00400 SYNC      CALL    212H            ;SELECT DRIVE
701F CD9602             00410           CALL    296H            ;SYNCRONIZE
7022 212B80             00420           LD      HL,DATA         ;STORE CODE HERE
7025 11923C             00430           LD      DE,3C00H+146
7028 CD3502             00440 FNH       CALL    235H
702B FE55               00450           CP      55H             ;FILE NAME HEADER?
702D 20F9               00460           JR      NZ,FNH
702F CD3502             00470 NAME      CALL    235H
7032 FE3C               00480           CP      3CH             ;DATA HEADER?
7034 280B               00490           JR      Z,BLKLTH
7036 12                 00500           LD      (DE),A          ;NAME TO VIDEO
7037 13                 00510           INC     DE
7038 18F5               00520           JR      NAME            ;GET FULL NAME
703A CD3502             00530 DRB       CALL    235H
703D FE3C               00540           CP      3CH             ;DATA HEADER?
703F 201F               00550           JR      NZ,CHECK
7041 CDEC70             00560 BLKLTH    CALL    GETBYT          ;GET/STORE BYTE
7044 47                 00570           LD      B,A             ;B=BLOCK LENGTH
7045 CDEC70             00580 LSB       CALL    GETBYT
7048 4F                 00590           LD      C,A             ;STORE BYTE IN C
7049 CDEC70             00600 MSB       CALL    GETBYT
704C 81                 00610           ADD     A,C             ;ADD THIS BYTE
704D 4F                 00620           LD      C,A
704E CDEC70             00630 BLOCK     CALL    GETBYT
7051 81                 00640           ADD     A,C
7052 4F                 00650           LD      C,A             ;CHECKSUM IN C
7053 10F9               00660           DJNZ    BLOCK           ;GET ALL OF BLOCK
7055 CD3502             00670 CHKSUM    CALL    235H
7058 B9                 00680           CP      C               ;CHECK CHECKSUM
7059 2057               00690           JR      NZ,ERROR1
705B CD2C02             00700           CALL    22CH            ;BLINK STARS
705E 18DA               00710           JR      DRB             ;LOOP
7060 FE78               00720 CHECK     CP      78H             ;END OF FILE?
7062 2053               00730           JR      NZ,ERROR2
7064 CD3502             00740           CALL    235H
7067 4F                 00750           LD      C,A             ;C=LSB OF ENTRY
7068 CD3502             00760           CALL    235H
706B 47                 00770           LD      B,A             ;B=MSB OF ENTRY
```

```
706C ED432980 00780          LD      (ENTRY),BC      ;STORE ENTRY ADDR.
7070 CDF801   00790          CALL    1F8H            ;STOP TAPE
7073 3600     00800          LD      (HL),0          ;STORE A ZERO
7075 23       00810          INC     HL
7076 3600     00820          LD      (HL),0          ;ANOTHER
7078 23       00830          INC     HL
7079 3600     00840          LD      (HL),0          ;THIRD
707B E5       00850          PUSH    HL              ;SAVE LAST ADDR.
707C 213871   00860          LD      HL,GETSPC       ;GET FILESPEC
707F CDCF44   00870          CALL    44CFH
              00880 ;-----------------------------------------------
              00890 ;        DATA READY...CREATE DUMP COMMAND
              00900 ;-----------------------------------------------
7082 060C     00910 FYLSPC   LD      B,12            ;FILESPEC LENGTH
7084 211D43   00920          LD      HL,431DH        ;PUT A COPY HERE
7087 CDD905   00930          CALL    5D9H            ;GET/DISPLAY DATA
708A 215071   00940          LD      HL,CMD2         ;" (START=X'
708D 111D43   00950          LD      DE,431DH
7090 7B       00960          LD      A,E             ;  8000',END=X'"
7091 80       00970          ADD     A,B             ;B=FILESPEC LENGTH
7092 5F       00980          LD      E,A
7093 011600   00990          LD      BC,22           ;# CHARS.
7096 EDB0     01000          LDIR                    ;MOVE CHARS.
7098 E1       01010          POP     HL              ;GET LAST ADDR.
7099 E5       01020          PUSH    HL              ;SAVE AGAIN
709A 7C       01030          LD      A,H             ;MSB
709B CDC370   01040          CALL    ASCII           ;DISPLAY AS ASCII
709E E1       01050          POP     HL              ;GET LAST ADDR.
709F 7D       01060          LD      A,L             ;LSB
70A0 CDC370   01070          CALL    ASCII           ;DISPLAY AS ASCII
70A3 216671   01080          LD      HL,CMD3         ;REST OF COMMAND
70A6 010F00   01090          LD      BC,15           ;# CHARS.
70A9 EDB0     01100          LDIR                    ;MOVE REMAINING
70AB 211843   01110          LD      HL,4318H        ;COMMAND LOCATION
70AE FB       01120          EI                      ;RESTART CLOCK
70AF C30544   01130          JP      4405H           ;EXECUTE COMMAND
              01140 ;-----------------------------------------------
              01150 ;        ROUTINES USED BY THIS PROGRAM
              01160 ;-----------------------------------------------
70B2 217671   01170 ERROR1   LD      HL,EM1          ;ERR MESSAGE ADDR.
70B5 1803     01180          JR      EINPUT
70B7 218571   01190 ERROR2   LD      HL,EM2          ;ERR MESSAGE ADDR.
70BA CDCF44   01200 EINPUT   CALL    44CFH           ;DISPLAY MESSAGE
70BD CDE470   01210          CALL    WAIT            ;AWAIT <ENTER>
70C0 C30070   01220          JP      LOADER          ;RESTART
70C3 4F       01230 ASCII    LD      C,A             ;SAVE BYTE
70C4 CB3F     01240          SRL     A               ;GET 4 HIGH BITS
70C6 CB3F     01250          SRL     A
70C8 CB3F     01260          SRL     A
70CA CB3F     01270          SRL     A
70CC CDDA70   01280          CALL    ATEST           ;CHANGE TO ASCII
70CF 12       01290          LD      (DE),A          ;DISPLAY
70D0 79       01300          LD      A,C             ;GET BYTE COPY
70D1 13       01310          INC     DE
70D2 E60F     01320          AND     0FH             ;GET 4 LOW BITS
70D4 CDDA70   01330          CALL    ATEST           ;CHANGE TO ASCII
70D7 12       01340          LD      (DE),A          ;DISPLAY
70D8 13       01350          INC     DE
70D9 C9       01360          RET
70DA C630     01370 ATEST    ADD     A,30H           ;TO GET ASCII
70DC FE3A     01380          CP      3AH
70DE FAE370   01390          JP      M,ATEST1
70E1 C607     01400          ADD     A,7             ;FOR "A" TO "F"
70E3 C9       01410 ATEST1   RET
70E4 CD4900   01420 WAIT     CALL    49H             ;GET KEY
70E7 FE0D     01430          CP      0DH             ;<ENTER>?
70E9 20F9     01440          JR      NZ,WAIT
70EB C9       01450          RET
70EC CD3502   01460 GETBYT   CALL    235H            ;BYTE FROM TAPE
70EF 77       01470          LD      (HL),A          ;STORE IT
```

Program continued

```
70F0 23        01480          INC     HL              ;NEXT LOCATION
70F1 C9        01490          RET
70F2 54        01500 TITLE    DEFM    'TAPE TO DISK TRANSFER UTILITY'
710F 0A        01510          DEFB    0AH
7110 52        01520          DEFM    'READY TAPE...<ENTER>'
7124 0A        01530          DEFB    0AH
7125 54        01540          DEFM    'TAPE FILES NAME :'
7136 0A03      01550          DEFW    30AH
7138 44        01560 GETSPC   DEFM    'DESIRED FILESPEC: '
714A 03        01570          DEFB    03H
714B 44        01580 CMD1     DEFM    'DUMP '
7150 20        01590 CMD2     DEFM    ' (START=X'
7159 27        01600          DEFB    27H                     ;"'"
715A 38        01610          DEFM    '8000'
715E 27        01620          DEFB    27H
715F 2C        01630          DEFM    ',END=X'
7165 27        01640          DEFB    027H
7166 27        01650 CMD3     DEFB    27H
7167 2C        01660          DEFM    ',TRA=X'
716D 27        01670          DEFB    27H
716E 38        01680          DEFM    '8000'
7172 2729      01690          DEFW    2927H
7174 200D      01700          DEFW    0D20H
7176 43        01710 EM1      DEFM    'CHECKSUM ERROR'
7184 0D        01720          DEFB    0DH
7185 45        01730 EM2      DEFM    'EOF ERROR'
718E 0D        01740          DEFB    0DH
               01750 ;-----------------------------------------------
               01760 ;        INITIALIZE LEVEL-2 BASIC VECTORS
               01770 ;        AND MOVE CODE TO INTENDED LOCATIONS
               01780 ;-----------------------------------------------
8000           01790          ORG     8000H
8000 F3        01800 MOVER    DI
8001 AF        01810          XOR     A
8002 21D206    01820          LD      HL,6D2H         ;VECTOR LOCATION
8005 110040    01830          LD      DE,4000H        ;DEST.
8008 013600    01840          LD      BC,36H          ;# BYTES
800B EDB0      01850          LDIR                    ;MOVE THEM
800D 212B80    01860          LD      HL,DATA         ;LOCATION OF DATA
8010 0600      01870 MOVER1   LD      B,0             ;CLEAR B
8012 4E        01880          LD      C,(HL)          ;BC=BLOCK LENGTH
8013 23        01890          INC     HL
8014 5E        01900          LD      E,(HL)          ;E=LSB OF DEST.
8015 23        01910          INC     HL
8016 56        01920          LD      D,(HL)          ;D=MSB OF DEST.
8017 23        01930          INC     HL
8018 7A        01940          LD      A,D             ;SEE IF D AND
8019 B3        01950          OR      E               ;E ARE ZERO
801A 280C      01960          JR      Z,XECUTE        ;IF SO,THEN DONE
801C 79        01970          LD      A,C             ;IF C=0 THEN BLOCK
801D FE00      01980          CP      0               ;LTH IS 256 BYTES
801F 2003      01990          JR      NZ,MCONT
8021 010001    02000          LD      BC,256
8024 EDB0      02010 MCONT    LDIR                    ;MOVE BLOCK
8026 18E8      02020          JR      MOVER1          ;LOOP
8028 C3        02030 XECUTE   DEFB    0C3H            ;"JP" CODE
8029 0000      02040 ENTRY    DEFW    0               ;ENTRY ADDR. HERE
802B 00        02050 DATA     NOP                     ;DATA LOCATION
7000           02060          END     LOADER
00000 TOTAL ERRORS
```

Generate

by Jim Rastin

Have you ever bought a piece of software on tape and then destroyed your only copy? Instead of wishing that you had a backup copy of that SYSTEM tape, read on to find out how you can back up almost any SYSTEM program you can buy.

GENER (see Program Listing), which is short for Generate, reads in a SYSTEM tape while doing a checksum of its contents. If an error occurs, the program stops the tape and displays a message on the screen. It then allows you to try loading at a different volume. If you change your mind while loading a program, press the / key which stops the tape and returns Generate to its start-up state. When your program is loaded, the screen displays the start, end, and entry points. At this time, the screen prompts you to press CLEAR to rename the resident program. If you choose to do this, the center of the screen clears and you can type in your new name, pressing the space bar to eliminate any extra letters. Pressing ENTER returns the program statistics to the screen with your program renamed.

If you now press ENTER, the screen clears, and a generating message appears. Again, you can press / at any time to abort the main program to the beginning. The resident program is not lost, and if you press @, it displays all the program statistics again. If you don't abort, when the program has finished generating, three options appear on the screen.

- Press N to restart for a new program.
- Press ENTER to generate another copy of the same program.
- Press / to return to BASIC.

One of the most interesting tricks in this program is in lines 3750 and 3760. If you place a new origin statement and then a byte of 0E9H (JP TO (HL)) at the end of any program, the system command automatically begins to execute your program without your having to press / and ENTER after it is loaded. I have listed this in my files as Auto Start.

Lines 160 to 370 of the program display messages to the screen. Lines 380–400 check for the @ key to see the resident program. Lines 430–990 load the program from tape. Lines 840 to 900 scan the keyboard for the / key to abort. Lines 1030 to 1560 calculate the program statistics and display them on the screen. Lines 1600 to 1670 stop the tape, display the error message, and then wait for you to press the ENTER key to return to the start of the program. Lines 1860 to 2450 generate the resident program to tape and then

scan the keyboard for a command. Lines 2500 to 2720 convert a hex number in the HL registers to ASCII and then display it where the DE registers are pointing. Lines 2740 to 3180 allow the user to rename the program in memory. Lines 3260 to 3320 display a message of any length to a point on the screen contained in the DE registers. The message must end with a 0. Lines 3380 to 3730 contain the different messages that the program uses to display on the screen.

You can change the message in line 3660 to anything you wish. Just be sure that your message contains fewer than 64 characters or that the 65th character starts a new word. Anything over 64 characters causes the message to wrap around to the next screen line. Your longest message must not exceed 128 characters.

If you use Auto Start, then Generate approximates the end of the resident program using the last load address from the tape to calculate the end address. In Auto Start, this address points to 41E2H which is wrong. Generate checks this and then returns the end point as the start point plus the number of bytes read (which includes addresses and pointers). That is why I say that the end address is approximate.

Program Listing. *Generate*

Encyclopedia Loader

```
                00100 ;* ================================= *
                00110 ;GENER SHORT FOR GENERATE WAS WRITTEN BY
                00120 ;       JIM RASTIN
                00130 ;       316 BRIARHILL AVE.
                00140 ;       LONDON ONTARIO  N5Y 1N8
                00150 ;* ================================= *
42E9            00160 START   EQU     42E9H
42E9            00170         ORG     START
42E9 31FF4F     00180         LD      SP,4FFFH
42EC CDC901     00190         CALL    01C9H           ;CLS
42EF 11183C     00200         LD      DE,3C18H
42F2 215447     00210         LD      HL,MES7         ;PROGRAM NAME
42F5 CD7545     00220         CALL    WRITE           ;WRITE MESSAGE TO SCREEN
42F8 11503C     00230         LD      DE,3C50H
42FB 217546     00240         LD      HL,MES1         ;TAPE LOAD ADDRESS 5000H
42FE CD7545     00250         CALL    WRITE           ;WRITE MESSAGE
4301 11803E     00260         LD      DE,3E80H
4304 21BE46     00270         LD      HL,MES3         ;PRESS @ TO SEE PROGRAM
4307 CD7545     00280         CALL    WRITE
430A 11003F     00290         LD      DE,3F00H
430D 216547     00300         LD      HL,MES8         ;COPYWRITE & NAME
4310 CD7545     00310         CALL    WRITE
4313 11C03E     00320         LD      DE,3EC0H
4316 21F845     00330         LD      HL,MESF         ;PRESS / TO ABORT
4319 CD7545     00340         CALL    WRITE
431C 11C03C     00350         LD      DE,3CC0H
431F 218E47     00360         LD      HL,MES9         ;PRESS <ENTER> TO START
4322 CD7545     00370         CALL    WRITE
4325 CD4900     00380 WAIT    CALL    0049H           ;KEY ENTRY
4328 FE40       00390         CP      40H             ;CHECK FOR @
432A CA6745     00400         JP      Z,RES           ;  IF YES RESIDENT
432D FE0D       00410         CP      0DH
432F 20F4       00420         JR      NZ,WAIT
4331 11C03C     00430         LD      DE,3CC0H
4334 214B46     00440         LD      HL,MESZ         ;PROG. LOADING P.S.B
4337 CD7545     00450         CALL    WRITE
433A 210050     00460         LD      HL,5000H        ;START
433D 11003D     00470         LD      DE,3D00H        ;SCREEN
4340 CD1202     00480         CALL    0212H           ;START TAPE
4343 CD9602     00490         CALL    0296H           ;FIND SYNC
4346 CD3502     00500         CALL    0235H           ;LOAD BYTE
4349 77         00510         LD      (HL),A
434A 23         00520         INC     HL
434B 0606       00530         LD      B,6
434D FE55       00540         CP      55H
434F C22C44     00550         JP      NZ,ERR
4352 CD3502     00560 LP1     CALL    0235H
4355 77         00570         LD      (HL),A          ;PUT NAME
4356 12         00580         LD      (DE),A          ;  OF PROGRAM
4357 23         00590         INC     HL              ;    ON SCREEN
4358 13         00600         INC     DE              ;       AND
4359 10F7       00610         DJNZ    LP1             ;    ON TAPE
435B 3E20       00620         LD      A,20H
435D 12         00630         LD      (DE),A
435E CD4244     00640 LP2     CALL    BLINK
4361 CD3502     00650         CALL    0235H
4364 77         00660         LD      (HL),A
4365 23         00670         INC     HL
4366 FE78       00680         CP      78H
4368 2835       00690         JR      Z,LAST          ;CHECK FOR END
436A CD3502     00700         CALL    0235H
436D 47         00710         LD      B,A
436E 77         00720         LD      (HL),A
436F 328045     00730         LD      (FINAL),A
4372 23         00740         INC     HL
4373 CD4B44     00750         CALL    LDHL            ;GET ADDRESS FROM TAPE
4376 ED538145   00760         LD      (FINAL+1),DE    ;STORE END -# BYTES
437A 83         00770         ADD     A,E             ;CALCULATE AND
437B 4F         00780         LD      C,A             ;STORE CHECKSUM
```

Program continued

```
437C CD3502   00790 LP3     CALL    0235H
437F 77       00800         LD      (HL),A          ;LOAD TO MEM.
4380 23       00810         INC     HL
4381 81       00820         ADD     A,C             ;CALCULATE AND
4382 4F       00830         LD      C,A             ;STORE CHECKSUM
4383 E5       00840         PUSH    HL              ;SCAN KEYBOARD
4384 C5       00850         PUSH    BC              ;   FOR
4385 CD2B00   00860         CALL    002BH           ;   / KEY
4388 FE2F     00870         CP      '/'             ;     TO ABORT
438A CAE744   00880         JP      Z,ABORT
438D C1       00890         POP     BC
438E E1       00900         POP     HL
438F 10EB     00910         DJNZ    LP3
4391 CD3502   00920         CALL    0235H
4394 77       00930         LD      (HL),A
4395 23       00940         INC     HL
4396 B9       00950         CP      C               ;CHECKSUM
4397 28C5     00960         JR      Z,LP2
4399 AF       00970         XOR     A
439A C32C44   00980         JP      ERR
0002          00990 STO1    DEFS    2
              01000 ;************************************
              01010 ;PROGRAM IN.  FIND START,END & ENTRY ADDRESSES
              01020 ;************************************
439F CD4B44   01030 LAST    CALL    LDHL
43A2 ED539D43 01040         LD      (STO1),DE
43A6 CDF801   01050         CALL    01F8H           ;STOP TAPE
43A9 23       01060         INC     HL
43AA 11FF4F   01070         LD      DE,4FFFH        ;START OF PROGRAM
43AD ED52     01080         SBC     HL,DE           ;DETERMINE LENGTH
43AF 227E45   01090         LD      (STORE),HL      ;STORE LENGTH
43B2 B7       01100         OR      A               ;RESET CARRY FLAG
43B3 3A8045   01110         LD      A,(FINAL)       ;GET # BYTES
43B6 2A8145   01120         LD      HL,(FINAL+1)
43B9 FE00     01130         CP      00H
43BB 2001     01140         JR      NZ,FINAL1
43BD 24       01150         INC     H
43BE 5F       01160 FINAL1  LD      E,A             ;PUT # BYTES IN DE
43BF 1600     01170         LD      D,0
43C1 19       01180         ADD     HL,DE           ;HL = END OF PROGRAM
43C2 228045   01190         LD      (FINAL),HL
              01200 ;*** CHECK FOR AUTO START ***
43C5 ED5B0950 01210         LD      DE,(5009H)      ;START ADDR
43C9 ED52     01220         SBC     HL,DE
43CB F2D943   01230         JP      P,REN1          ;GO IF END > START
43CE 2A7E45   01240         LD      HL,(STORE)      ;APPROX. LENGTH
43D1 ED5B0950 01250         LD      DE,(5009H)      ;START
43D5 19       01260         ADD     HL,DE           ;APPROX. END
43D6 228045   01270         LD      (FINAL),HL
43D9 21803E   01280 REN1    LD      HL,3E80H        ;ERASE
43DC 11813E   01290         LD      DE,3E81H        ; RESIDENT
43DF 012A00   01300         LD      BC,42           ;   PROGRAM
43E2 3620     01310         LD      (HL),20H        ;       MESSAGE
43E4 EDB0     01320         LDIR
43E6 21C03C   01330         LD      HL,3CC0H        ; ERASE
43E9 11C13C   01340         LD      DE,3CC1H        ;  LOADING
43EC 012C00   01350         LD      BC,44           ;    MESSAGE
43EF 3620     01360         LD      (HL),20H
43F1 EDB0     01370         LDIR
43F3 11803E   01380         LD      DE,3E80H
43F6 218345   01390         LD      HL,MESA         ;PRESS <ENTER> GEN.
43F9 CD7545   01400         CALL    WRITE
43FC 11803D   01410         LD      DE,3D80H
43FF 21CE45   01420         LD      HL,MESC         ;START POINT
4402 CD7545   01430         CALL    WRITE
4405 2A0950   01440         LD      HL,(5009H)
4408 CDED44   01450         CALL    CONV
440B 11C03D   01460         LD      DE,3DC0H
440E 21DC45   01470         LD      HL,MESD         ;ENTRY POINT
4411 CD7545   01480         CALL    WRITE
4414 2A9D43   01490         LD      HL,(STO1)
4417 CDED44   01500         CALL    CONV
```

```
441A 11003E   01510          LD      DE,3E00H
441D 21EA45   01520          LD      HL,MESE          ;END POINT
4420 CD7545   01530          CALL    WRITE
4423 2A8045   01540          LD      HL,(FINAL)
4426 CDED44   01550          CALL    CONV
4429 C35844   01560          JP      LOAD
              01570 ;*****************************
              01580 ;CHECKSUM ERROR ROUTINE
              01590 ;*****************************
442C CDF801   01600 ERR      CALL    01F8H            ;STOP RECORDER
442F 11403D   01610          LD      DE,3D40H
4432 219646   01620          LD      HL,MES2          ;CHECKSUM ERROR
4435 CD7545   01630          CALL    WRITE
4438 CD4900   01640 LPL1     CALL    0049H            ;KEY ENTRY
443B FE0D     01650          CP      0DH
443D 20F9     01660          JR      NZ,LPL1
443F C3E942   01670          JP      START
              01680 ;**** BLINK STAR IN UPPER RIGHT ****
4442 3A3F3C   01690 BLINK    LD      A,(3C3FH)
4445 EE0A     01700          XOR     0AH
4447 323F3C   01710          LD      (3C3FH),A
444A C9       01720          RET
              01730 ;**** LD 2 BYTES TO WHERE HL POINTS TO ****
444B CD3502   01740 LDHL     CALL    0235H
444E 5F       01750          LD      E,A
444F 77       01760          LD      (HL),A
4450 23       01770          INC     HL
4451 CD3502   01780          CALL    0235H
4454 57       01790          LD      D,A
4455 77       01800          LD      (HL),A
4456 23       01810          INC     HL
4457 C9       01820          RET
              01830 ;*****************************
              01840 ;ROUTINE TO WRITE PROGRAM TO TAPE
              01850 ;*****************************
4458 11403E   01860 LOAD     LD      DE,3E40H
445B 21A645   01870          LD      HL,MESB          ;PRESS <CLEAR> TO RENAME
445E CD7545   01880          CALL    WRITE
4461 11803E   01890          LD      DE,3E80H
4464 218345   01900          LD      HL,MESA          ;PRESS <ENTER> TO GEN.
4467 CD7545   01910          CALL    WRITE
446A CD4900   01920          CALL    0049H            ;SCAN  KEYS
446D FE1F     01930          CP      1FH              ; CHECK FOR <CLEAR>
446F CA0F45   01940          JP      Z,NAME           ;   IF YES GOTO NAME
4472 FE2F     01950          CP      '/'              ;   CHECK FOR
4474 CAE744   01960          JP      Z,ABORT          ;     SLASH  KEY
4477 FE0D     01970          CP      0DH              ;CHECK FOR <ENTER>
4479 20DD     01980          JR      NZ,LOAD
447B CDC901   01990 LPL2     CALL    01C9H            ;CLS
447E 11803D   02000          LD      DE,3D80H
4481 212647   02010          LD      HL,MES5          ;PROG. GENERATING P.S.B
4484 CD7545   02020          CALL    WRITE
4487 CD1202   02030          CALL    0212H            ;START TAPE
448A CD8702   02040          CALL    0287H            ;WRITE LEADER
448D 210050   02050          LD      HL,5000H
4490 ED4B7E45 02060          LD      BC,(STORE)       ;GET LENGTH OF PROGRAM
4494 7E       02070 LD1      LD      A,(HL)
4495 CD6402   02080          CALL    0264H            ;OUTPUT BYTE
4498 23       02090          INC     HL
4499 0B       02100          DEC     BC
449A 78       02110          LD      A,B
449B B1       02120          OR      C
449C 2814     02130          JR      Z,STP
449E D9       02140          EXX                      ;FLASH
449F 05       02150          DEC     B                ;  STAR ON SCrEEN
44A0 CC4244   02160          CALL    Z,BLINK          ;     EVERY 256
44A3 D9       02170          EXX                      ;       BYTES
44A4 E5       02180          PUSH    HL
44A5 C5       02190          PUSH    BC
44A6 CD2B00   02200          CALL    002BH            ;CHECK
44A9 FE2F     02210          CP      '/'              ; FOR
```

Program continued

```
44AB CAE744   02220          JP      Z,ABORT          ;   ABORT
44AE C1       02230          POP     BC               ;     COMMAND
44AF E1       02240          POP     HL
44B0 20E2     02250          JR      NZ,LD1
44B2 CDF801   02260 STP      CALL    01F8H            ;STOP DRIVE
44B5 CDC901   02270          CALL    01C9H            ;CLS
44B8 11003D   02280          LD      DE,3D00H
44BB 21ED46   02290          LD      HL,MES4          ;<ENTER> FOR MORE COPIES
44BE CD7545   02300          CALL    WRITE
44C1 11403D   02310          LD      DE,3D40H
44C4 21B847   02320          LD      HL,MES10         ;/ TO GOTO BASIC
44C7 CD7545   02330          CALL    WRITE
44CA 11803D   02340          LD      DE,3D80H
44CD 21F447   02350          LD      HL,MES11         ;  <N> TO RESTART
44D0 CD7545   02360          CALL    WRITE
              02370 ;*** SCAN KEYS FOR COMMAND ***
44D3 CD4900   02380 SCAN     CALL    0049H            ;KEY BOARD
44D6 FE4E     02390          CP      4EH
44D8 CAE942   02400          JP      Z,START
44DB FE0D     02410          CP      0DH
44DD CA7B44   02420          JP      Z,LPL2
44E0 FE2F     02430          CP      '/'
44E2 CA0000   02440          JP      Z,0000H
44E5 18EC     02450          JR      SCAN
              02460 ;**** RESTART ROUTINE ****
44E7 CDF801   02470 ABORT    CALL    01F8H            ;ABORT PROGRAM
44EA C3E942   02480          JP      START            ;     RETURN TO  START
              02490 ;*** CONVERT HL TO ASCII ***
44ED 13       02500 CONV     INC     DE
44EE 7C       02510          LD      A,H
44EF CDF744   02520          CALL    COV1
44F2 7D       02530          LD      A,L
44F3 CDF744   02540          CALL    COV1
44F6 C9       02550          RET
44F7 F5       02560 COV1     PUSH    AF
44F8 0602     02570          LD      B,2
44FA 0F       02580          RRCA                     ;CONVERT
44FB 0F       02590          RRCA                     ;   HEX
44FC 0F       02600          RRCA                     ;     TO
44FD 0F       02610          RRCA                     ;       ASCII
44FE E60F     02620 COV2     AND     0FH              ;MASK  1ST CHAR.
4500 FE0A     02630          CP      0AH              ;  CHECK FOR NUMBER
4502 3802     02640          JR      C,NUM2
4504 C607     02650          ADD     A,7
4506 C630     02660 NUM2     ADD     A,30H            ;CONV TO ASCII
4508 12       02670          LD      (DE),A
4509 13       02680          INC     DE
450A 05       02690          DEC     B
450B C8       02700          RET     Z
450C F1       02710          POP     AF
450D 18EF     02720          JR      COV2
              02730 ;*** CHANGE NAME ROUTINE ***
450F 21803D   02740 NAME     LD      HL,3D80H         ;CLEAR PART
4512 11813D   02750          LD      DE,3D81H         ;  OF
4515 01FF00   02760          LD      BC,0FFH          ;   SCREEN
4518 3620     02770          LD      (HL),20H
451A EDB0     02780          LDIR
451C 11803D   02790          LD      DE,3D80H
451F 212346   02800          LD      HL,MESH          ;TYPE NAME & <ENTER>
4522 CD7545   02810          CALL    WRITE
4525 21003D   02820          LD      HL,3D00H
4528 110150   02830          LD      DE,5001H
452B 0607     02840          LD      B,7
452D E5       02850 NM2      PUSH    HL
452E D5       02860          PUSH    DE
452F C5       02870          PUSH    BC
4530 CD4900   02880          CALL    0049H            ;SCAN KEYS
4533 C1       02890          POP     BC
4534 D1       02900          POP     DE
4535 E1       02910          POP     HL
4536 FE08     02920          CP      08H              ;  CHECK FOR BACK ARROW
4538 280E     02930          JR      Z,BACK
```

```
453A FE0D     02940          CP      0DH             ;CHECK FOR ENTER
453C CA5745   02950          JP      Z,REN3
453F 05       02960          DEC     B
4540 2812     02970          JR      Z,REN2
4542 77       02980          LD      (HL),A
4543 12       02990          LD      (DE),A
4544 23       03000          INC     HL
4545 13       03010          INC     DE
4546 18E5     03020          JR      NM2             ;GET NEXT CHAR.
4548 7D       03030 BACK     LD      A,L
4549 FE00     03040          CP      00H
454B 28C2     03050          JR      Z,NAME
454D 2B       03060          DEC     HL
454E 3620     03070          LD      (HL),20H
4550 1B       03080          DEC     DE
4551 04       03090          INC     B
4552 18D9     03100          JR      NM2
4554 04       03110 REN2     INC     B
4555 18D6     03120          JR      NM2
4557 21803D   03130 REN3     LD      HL,3D80H        ;ERASE
455A 11813D   03140          LD      DE,3D81H        ; RENAME
455D 01FF00   03150          LD      BC,0FFH         ;    MESSAGE
4560 3620     03160          LD      (HL),20H
4562 EDB0     03170          LDIR
4564 C3D943   03180          JP      REN1
              03190 ;*** NAME TO SCREEN ***
4567 11003D   03200 RES      LD      DE,3D00H
456A 210150   03210          LD      HL,5001H
456D 010600   03220          LD      BC,6
4570 EDB0     03230          LDIR
4572 C3D943   03240          JP      REN1            ;JP TO  RENAME 1
              03250 ;*** WRITE MESSAGE TO SCREEN ***
4575 7E       03260 WRITE    LD      A,(HL)
4576 FE00     03270          CP      0
4578 C8       03280          RET     Z
4579 12       03290          LD      (DE),A
457A 13       03300          INC     DE
457B 23       03310          INC     HL
457C 18F7     03320          JR      WRITE
0002          03330 STORE    DEFS    2
0003          03340 FINAL    DEFS    3
              03350 ;***********************************
              03360 ;TABLE OF MESSAGES (MUST END IN 0)
              03370 ;***********************************
4583 2A       03380 MESA     DEFM    '** PRESS <ENTER> TO GENERATE **   '
45A5 00       03390          DEFB    0
45A6 2A       03400 MESB     DEFM    '** PRESS <CLEAR> TO RENAME PROGRAM **  '
45CD 00       03410          DEFB    0
45CE 53       03420 MESC     DEFM    'START POINT  '
45DB 00       03430          DEFB    0
45DC 45       03440 MESD     DEFM    'ENTRY POINT  '
45E9 00       03450          DEFB    0
45EA 45       03460 MESE     DEFM    'END   POINT  '
45F7 00       03470          DEFB    0
45F8 2A       03480 MESF     DEFM    '** PRESS <SLASH> TO ABORT AT ANY TIME **  '
4622 00       03490          DEFB    0
4623 2A       03500 MESH     DEFM    '** TYPE NEW NAME AND PRESS <ENTER> **  '
464A 00       03510          DEFB    0
464B 2A       03520 MESZ     DEFM    '** PROGRAM  LOADING  PLEASE STAND BY **  '
4674 00       03530          DEFB    0
4675 54       03540 MES1     DEFM    'TAPE LOADED AT LOCATION 5000H   '
4695 00       03550          DEFB    0
4696 43       03560 MES2     DEFM    'CHECKSUM ERROR - CHECK VOL.&TRY AGAIN  '
46BD 00       03570          DEFB    0
46BE 2A       03580 MES3     DEFM    '** PRESS <@> TO SEE RESIDENT PROGRAM  **      '
46EC 00       03590          DEFB    0
46ED 2A       03600 MES4     DEFM    '** PRESS <ENTER> TO GENERATE MORE COPIES **
4725 00       03610          DEFB    0
4726 2A       03620 MES5     DEFM    '** PROGRAM GENERATING PLEASE STAND BY **     '
4753 00       03630          DEFB    0
4754 2A       03640 MES7     DEFM    '** GENERATE **  '
```

Program continued

```
4764 00        03650          DEFB    0
4765 2A        03660 MES8     DEFM    '** WRITTEN BY JIM RASTIN FOR THE      **
                                        ** ENCYCLOPEDIA OF THE TRS-80         **'
478D 00        03670          DEFB    0
478E 2A        03680 MES9     DEFM    '** PRESS <ENTER> TO START WHEN READY **  '
47B7 00        03690          DEFB    0
47B8 2A        03700 MES10    DEFM    '** PRESS <SLASH> TO RETURN TO BASIC    **
47F3 00        03710          DEFB    0
47F4 2A        03720 MES11    DEFM    '** PRESS <N> TO GENERATE A NEW PROGRAM **
482E 00        03730          DEFB    0
               03740 ;*** ROUTINE TO AUTO START SYSTEM PROGRAM ***
41E2           03750          ORG     41E2H
41E2 E9        03760          DEFB    0E9H    ;JP TO (HL)
42E9           03770          END     START
00000 TOTAL ERRORS
```

Professional Looking Listings with a Teletype™

by Fred M. Conover

Those of you who use a Teletype printer probably have experienced the same problems that I have when listing a program. First, if you have program lines which exceed the capacity of the machine (normally around 75 characters per line), one of two things happens: If you do not have automatic carriage return, all of the extra characters pile up at the right side, making them unreadable; if you do have automatic carriage return, then one or more characters is printed randomly across the page as the carriage returns. Again, this makes the listing hard to read and very nonprofessional looking. Second, if you keep your listings neatly bound in listing binders, and the lines of code you are interested in are under the binder, you cannot read them.

The Program Listing is an assembly-language program that will solve these problems. The program will output 75 characters (or any number you desire) to the printer and output a line feed/carriage return. It will then indent five spaces and continue with the rest of the line. It will also print 50 lines and turn up a new page, all automatically. If you use fan-fold paper, which holds 66 lines of printing, the program will subtract the number of lines you want per page (set for 50, easily changed) from 66 and send the correct number of line feeds to the printer to turn up a new page.

The printer driver routine resides in ROM and cannot be changed. Fortunately, the address of the driver routine is stored in RAM and can be changed to the address of a new printer driver routine. The *Level II BASIC Manual* memory map shows that the address of the driver routine is kept at 4026-4027H (16422-16423). You must change this address. I've located my new driver routine at 7F80H (32640), so this address must be placed in 4026-4027H. Lines 1000-1020 accomplish this. The ADD A,B instruction places an 80 in 4026H, and the LD A,A places a 7F in 4027H. Now when the computer wants to output a character to the printer it will jump to address 7F80 to find the driver routine. The ORG statement at line 1030 sets the origin for the rest of the routine at 7F80H.

The routine counts each character as it is printed and compares the number with the number in line 1160. When the numbers are equal, the routine outputs a carriage return/line feed, indents five spaces, and continues printing the rest of the line. If you want a different number of characters per line, change the number in line 1160. Note: If you are using

T-BUG to enter this routine, the number you enter must be in hex. As an example, if you want 70 characters per line, the hex code you enter at line 1160 would be FE46H.

The routine also counts all carriage returns, whether they are generated by the computer or generated by this routine, and compares the number to the number in line 1270. (Change this number to the number of lines per page you want as outlined above.) When they are equal, the routine generates the number of line feeds needed to turn up a new page to the same position as on the first page. The routine always checks to see if your printer is ready to print before outputting a character. Lines 1210–1230 initiate a 2.3 millisecond delay after each carriage return/line feed. This allows a 60 words-per-minute machine carriage about 10 percent more time to return before the next character is printed.

The routine can be entered by either T-BUG or by the Editor/Assembler program. Once you have the routine on tape you are very close to having professional looking listings. Set memory size to 32640. Load this routine using SYSTEM. When the ? prompt appears, hit the BREAK key to return to BASIC. Now load your program to be LLISTed in the usual manner and then LLIST. Your listing will have exactly 50 lines per page and never more than 75 characters per line.

When you have finished LLISTing and want to run your program without using this driver routine, you can restore the original driver routine by POKEing the following to replace the original driver address in 4026–4027H: POKE 16422,141 - POKE 16423,5.

If you have 32K RAM, change line 1020 to CP A and line 1030 to ORG 0BF80H. Set memory size to 49024. If you have 48K RAM, change line 1020 to RST 38H and line 1030 to ORG 0FF80H. Set memory size to 65408. This will locate the routine to the top of memory in each case. The routine uses 127 bytes.

Program Listing

```
4026            01000           ORG     4026H   ;LPRINT DRIVER ROUTINE
4026 80         01010           ADD     A,B
4027 7F         01020           LD      A,A     ;LD 4026-27 WITH NEW ADD.
7F80            01030           ORG     7F80H   ;NEW DRIVER ADDRESS
7F80 79         01040           LD      A,C     ;C = CHARACTER TO PRINT
7F81 B7         01050           OR      A       ;IS CHAR = 0
7F82 C8         01060           RET     Z       ;YES - RETURN
7F83 DDE5       01070           PUSH    IX      ;SAVE IX
7F85 DD21FD7F   01080           LD      IX,LINES        ;3 BYTE STORAGE AREA
7F89 DDBE02     01090           CP      (IX+2)  ;A - FFH
7F8C 3842       01100           JR      C,UPDATE        ;JUMP IF FLAG SET
7F8E FE0D       01110           CP      0DH     ;CARRIAGE RET. / LF
7F90 2811       01120           JR      Z,CRLF
7F92 CDEE7F     01130 THRU      CALL    PRINT
7F95 DD3401     01140           INC     (IX+1)  ;INC CHARACTER COUNTER
7F98 DD7E01     01150           LD      A,(IX+1);CHARACTER COUNTER
7F9B FE4B       01160           CP      75      ;CHARACTERS PER LINE
7F9D 202D       01170           JR      NZ,OUT  ;JUMP NOT 75
7F9F DD3602FF   01180           LD      (IX+2),0FFH     ;SET FLAG
7FA3 3E0D       01190 CRLF      LD      A,0DH   ;CR/LF
7FA5 CDEE7F     01200           CALL    PRINT
7FA8 0600       01210           LD      B,0     ;SET UP DELAY
7FAA 05         01220 LOOP3     DEC     B       ;2.3 MS DELAY
7FAB 20FD       01230           JR      NZ,LOOP3
7FAD DD360100   01240           LD      (IX+1),0        ;RESET CHARACTER COUNTER
7FB1 DD3400     01250           INC     (IX)    ;INC LINE COUNTER
7FB4 DD7E00     01260           LD      A,(IX)  ;LINE COUNTER
7FB7 FE32       01270           CP      50      ;IS IT 50 LINES
7FB9 2011       01280           JR      NZ,OUT  ;RETURN IF NOT 50
7FBB 3E42       01290           LD      A,66    ;LINES PER PAGE = 66
7FBD DD9600     01300           SUB     (IX)    ;PRINTED LINES
7FC0 47         01310           LD      B,A     ;NO. OF LF'S NEEDED
7FC1 3E0D       01320           LD      A,0DH   ;CR/LF
7FC3 CDEE7F     01330 LOOP      CALL    PRINT
7FC6 10FB       01340           DJNZ    LOOP    ;"B" TIMES
7FC8 DD360000   01350           LD      (IX),0  ;RESET LINE COUNTER
7FCC 79         01360 OUT       LD      A,C     ;RESTORE CHARACTER
7FCD DDE1       01370           POP     IX      ;RESTORE IX
7FCF C9         01380           RET             ;RETURN TO CALLING PROGRAM
7FD0 FE0D       01390 UPDATE    CP      0DH     ;IS IT CR/LF
7FD2 2006       01400           JR      NZ,INDENT       ;JUMP IF NOT CR/LF
7FD4 DD360200   01410           LD      (IX+2),0        ;RESET FLAG
7FD8 18F2       01420           JR      OUT
7FDA 0605       01430 INDENT    LD      B,5     ;INDENT 5 SPACES
7FDC 3E20       01440           LD      A,20H   ;SPACE
7FDE CDEE7F     01450 LOOP2     CALL    PRINT
7FE1 10FB       01460           DJNZ    LOOP2   ;5 TIMES
7FE3 DD360105   01470           LD      (IX+1),5        ;PRESET CHAR. COUNTER
7FE7 DD360200   01480           LD      (IX+2),0        ;RESET FLAG
7FEB 79         01490           LD      A,C     ;RESTORE CHAR.
7FEC 18A4       01500           JR      THRU
7FEE F5         01510 PRINT     PUSH    AF      ;SAVE CHAR.
7FEF 3AE837     01520 LOOP1     LD      A,(37E8H)       ;READ PRINTER
7FF2 E6F0       01530           AND     0F0H    ;MASK 4 LOWER BITS
7FF4 FE30       01540           CP      30H     ;30H = READY
7FF6 20F7       01550           JR      NZ,LOOP1;NOT READY - DO AGAIN
7FF8 F1         01560           POP     AF      ;RESTORE CHARACTER
7FF9 32E837     01570           LD      (37E8H),A       ;PRINT CHAR.
7FFC C9         01580           RET
7FFD 0000       01590 LINES     DEFW    00      ;STORAGE AREA
7FFF 00         01600           DEFB    0
0000            01610           END
00000 TOTAL ERRORS
```

More Patches to EDTASM

by Warren A. Smethurst

Due to memory size constraints, Radio Shack's TRS-80 Editor/Assembler (EDTASM) 1.2 for the 16K Model I has several limitations. There is no global search/replace for character strings, no macro support, and there are no page formatting directives for hard-copy output. It also lacks the ability to move lines of source code around in the internal text buffer, which makes it difficult to compose programs on-line. This article describes a routine you can patch into EDTASM to provide for moving lines of source code.

To apply the patches described here and in the source code listing, you have to use a monitor such as Radio Shack's T-BUG. If you have T-BUG, you must move it higher in memory to prevent conflict with EDTASM (4300H–5CDBH). See "Get T-BUG High" by Irwin Rappaport, *80 Microcomputing*, January 1980, p. 118. The easiest approach to implementing these patches is to use your current version of EDTASM to create a source file of the patches, then assemble it, saving it to tape. Using the SYSTEM command, load EDTASM, then the patches. While still in the SYSTEM mode, load your monitor and PUNCH 4300H through 5CDBH with starting address 468AH.

Patching EDTASM to use the ROM video device control block instead of its own gives you enough area (4460H thru 45A9H) to allow the insertion of the MOVE routine. See "Custom EDTASM" by John T. Blair, *80 Microcomputing*, August 1980, p. 122.

Another patch replaces the B command code (return to BASIC) with new commands M and MD (move and move/delete). Once this is done, the only way to exit EDTASM is to press the RESET button or turn off the machine.

If you want to replace EDTASM's REPLACE command processing with a global search/replace, the EDTASM REPLACE logic consists of only a handful of instructions that find the line, build return linkage to the INSERT routine, and branch into the DELETE routine.

The MOVE routine finds the first line to be moved and places it into the input buffer at 4127H as though it had just been keyed. Then, utilizing the EDTASM INSERT routine, it expands the text buffer and adds the line. If you select the DELETE option, the original line is deleted. The routine then goes back to see if there are any more lines to move. Using EDTASM's own line number validation and text manipulation routines assures protection against overlapping line numbers or expansion beyond the end of memory.

The M and MD command formats are explained in the documentation comments at the beginning of the source code listing. The M or MD command codes with no parameters utilize the line number parameters supplied from the previous use of the MOVE command. This allows successive lines to be moved one at a time in the event that not enough were moved the first time. If you issue the M or MD commands with no text in the buffer, you get the BAD PARAMETER(S) or NO SUCH LINE messages instead of NO TEXT IN BUFFER because the original B command code logic bypasses that test.

Notice that there is a special patch at address 43F0H for the keyboard driver. This is because EDTASM's keyboard driver stores data at address 4021H which is in the middle of the ROM video DCB. The patch changes the address to 4036H which is the one used by the ROM keyboard driver. Immediately following this patch in the source code listing, I've superimposed Radio Shack's KBFIX code over EDTASM's keyboard driver to provide a built-in fix for keyboard bounce.

Program Listing. *Patches to EDTASM 1.2*


```
                00010
                00020 ; PATCHES TO TRS-80 EDITOR/ASSEMBLER 1.2
                00030
                00040 ; WRITTEN BY:   W. SMETHURST
                00050
                00060 ; DATE WRITTEN: AUG. 8, 1981
                00070
                00080
                00090
                00100 ; ROUTINE DESCRIPTION:
                00110
                00120 ; THIS IS A ROUTINE TO MOVE ONE OR MORE LINES WITHIN THE
                00130 ; EDTASM TEXT BUFFER WITH AN OPTIONAL "DELETE AFTER MOVE"
                00140 ; CAPABILITY.
                00150
                00160 ; INCLUDED WITH THIS ROUTINE ARE THE "ORG" PATCHES TO
                00170 ; MODIFY EDTASM TO USE THE ROM VIDEO DCB AND BRANCH TO
                00180 ; THIS ROUTINE WITH THE NEW "M" COMMAND CODE.
                00190
                00200 ; THE ENTIRE ROUTINE OCCUPIES THE AREA USED BY EDTASM
                00210 ; FOR ITS VIDEO DRIVER LOGIC AND THE "M" COMMAND
                00220 ; REPLACES THE CURRENT "B" COMMAND (RETURN TO BASIC).
                00230
                00240 ; TO EXIT FROM EDTASM WILL NOW REQUIRE PRESSING THE RESET
                00250 ; BUTTON OR TURNING OFF THE MACHINE.
                00260
                00270 ; COMMAND ENTRY DESCRIPTION:
                00280
                00290 ;       M 10,50:100,1 - WILL MOVE LINES 10-50 FROM THEIR
                00300 ;                       CURRENT POSITION AND INSERT THEM
                00310 ;                       AT OR AFTER LINE 100 INCREMENTING
                00320 ;                       EACH LINE NO. BY ONE
                00330
                00340 ;       M 10:100,1    - WILL MOVE LINE 10 FROM ITS
                00350 ;                       CURRENT POSITION AND INSERT IT
                00360 ;                       AT OR AFTER LINE 100
                00370
                00380 ;       M 10:100      - WILL MOVE LINE 10 AS ABOVE AND
                00390 ;                       USE THE CURRENT EDTASM LINE NO.
                00400 ;                       INCREMENT
                00410
                00420 ;       M             - WILL MOVE THE LINE FOLLOWING THE
                00430 ;                       LAST ONE MOVED TO A POSITION
                00440 ;                       FOLLOWING WHERE THE LAST ONE WAS
                00450 ;                       INSERTED USING THE DEFAULT EDTASM
                00460 ;                       INCREMENT
                00470
                00480 ; DELETE OPTION:
                00490
                00500 ; BY PLACING THE LETTER "D" IMMEDIATELY AFTER THE "M"
                00510 ; COMMAND (MAKING "MD"), THE DELETE OPTION WILL BE
                00520 ; ACTIVATED AND THE LINES MOVED WILL BE DELETED FROM
                00530 ; THEIR ORIGINAL POSITIONS (EX:  MD 10,50:100  WILL
                00540 ; DELETE LINES 10 THRU 50 AFTER MOVING THEM).
                00550
                00560 ; SPECIAL PATCH TO KEYBOARD DRIVER REQUIRED WHEN USING
                00570 ; ROM VIDEO DCB:
                00580
43F0            00590         ORG     43F0H
43F0 3640       00600         DEFW    4036H
                00610
4407            00620         ORG     4407H
                00630
                00640 ; KEYBOARD DEBOUNCE PATCHES FROM KBFIX:
                00650
4407 C5         00660         PUSH    BC
4408 01DC05     00670         LD      BC,5DCH
```

```
440B CD6000   00680          CALL    60H              ; DELAY
440E C1       00690          POP     BC
440F 0A       00700          LD      A,(BC)
4410 A3       00710          AND     E
4411 C8       00720          RET     Z
4412 C3FB03   00730          JP      3FBH             ; CONTINUE WITH ROM DRIVER
              00740
4460          00750          ORG     4460H    ; EDTASM VIDEO DRIVER ADDRESS
              00760
              00770 ; EQUATES FOR EDTASM STORAGE ADDRESSES:
              00780
4111          00790 CURPTR   EQU     4111H            ; CURRENT LINE ADDRESS
4117          00800 INCR     EQU     4117H            ; DEFAULT LINE INCREMENT
411B          00810 LINPTR   EQU     411BH            ; ADDRESS FROM FNDLIN
411D          00820 NXTPTR   EQU     411DH            ; NXT ADDRESS FROM FNDLIN
4127          00830 BUFFER   EQU     4127H            ; 128 CHAR INPUT BUFFER
41A8          00840 BUFPTR   EQU     41A8H            ; POINTER TO INPUT BUFFER
41AA          00850 BUFLTH   EQU     41AAH            ; LENGTH OF INPUT DATA
              00860
              00870 ; EQUATES FOR EDTASM ROUTINES AND ENTRY POINTS:
              00880
4737          00890 LINEFD   EQU     4737H            ; DOES CRT LINE FEED
4941          00900 GETIPT   EQU     4941H            ; NORMAL INPUT ROUTINE
4A5C          00910 INSERT   EQU     4A5CH            ; ADD LINE TO TEXT BUFFER
4A83          00920 PROGSW   EQU     4A83H            ; ADDR OF CALL TO GETIPT
4AC1          00930 MOVE     EQU     4AC1H            ; ENTRY TO LDIR ROUTINE
4AC7          00940 LSTPRM   EQU     4AC7H            ; GET FINAL CMD PARAMS.
4ADA          00950 FNDLIN   EQU     4ADAH            ; FIND ON LINE NO.
4B09          00960 GETPRM   EQU     4B09H            ; GET INITIAL CMD PARAMS.
4B16          00970 GETLNO   EQU     4B16H            ; LOAD HL WITH LINE NO.
4BC2          00980 CPHLDE   EQU     4BC2H            ; COMPARE HL AND DE REGS
4CEA          00990 DELETE   EQU     4CEAH            ; ENTRY TO DELETE ROUTINE
4E29          01000 DISPLY   EQU     4E29H            ; CRT LINE DISPLAY
              01010
              01020 ; EDTASM ERROR MESSAGE ROUTINES:
              01030
4A3B          01040 BADNO    EQU     4A3BH            ; "LINE NO. TOO LARGE"
4B24          01050 BADPRM   EQU     4B24H            ; "BAD PARAMETER(S)"
4CBB          01060 NOLINE   EQU     4CBBH            ; "NO SUCH LINE"
              01070
              01080 ; "MOVE" ROUTINE STORAGE AREAS:
              01090
4460 0000     01100 CURLIN   DEFW    0                ; STARTING LINE NO.
4462 0000     01110 LSTLIN   DEFW    0                ; ENDING LINE NO.
4464 0000     01120 OLDLIN   DEFW    0                ; SAVE LINE NO. FOR DELETE
4466 00       01130 SWITCH   DEFB    0                ; DELETE OPTION SWITCH
              01140
4467          01150 START    EQU     $
              01160
4467 AF       01170          XOR     A
4468 326644   01180          LD      (SWITCH),A       ; ZERO = NO DELETE
446B 2A6044   01190          LD      HL,(CURLIN)
446E 226244   01200          LD      (LSTLIN),HL
              01210
              01220 ; CHECK TO SEE IF THIS IS ONLY THE "M" COMMAND WITH NO
              01230 ; PARAMETERS.
              01240
4471 3AAA41   01250          LD      A,(BUFLTH)       ; REMAINING COMMAND CHARS
4474 B7       01260          OR      A
4475 2817     01270          JR      Z,NOPRMS         ; ONLY "M" ENTERED
              01280
              01290 ; LOOK AT THE NEXT POSITION OF THE COMMAND ENTRY TO SEE
              01300 ; IF THE DELETE OPTION IS DESIRED.
              01310
4477 47       01320          LD      B,A
4478 2AA841   01330          LD      HL,(BUFPTR)
447B 7E       01340          LD      A,(HL)
447C FE44     01350          CP      'D'              ; DELETE?
447E 201B     01360          JR      NZ,FNDARG        ; NO
              01370
```

Program continued

```
                01380 ; DELETE OPTION IS DESIRED, SET THE SWITCH, ADJUST THE
                01390 ; REMAINING CHARACTERS AND COMMAND POINTER.
                01400
4480 326644     01410           LD      (SWITCH),A      ; NON-ZERO = DELETE
4483 23         01420           INC     HL              ; ADJUST STRING POINTER
4484 22A841     01430           LD      (BUFPTR),HL     ;   TO SKIP OVER THE "D".
4487 05         01440           DEC     B               ; REDUCE CHARS. REMAINING
4488 78         01450           LD      A,B
4489 32AA41     01460           LD      (BUFLTH),A
448C 200D       01470           JR      NZ,FNDARG
                01480
448E            01490 NOPRMS    EQU     $
                01500
                01510 ; IF NO PARAMETERS ENTERED, MAKE SURE THERE IS A CURRENT
                01520 ; LINE TO MOVE.
                01530
448E 2A6044     01540           LD      HL,(CURLIN)     ; NEXT LINE TO BE MOVED
4491 7C         01550           LD      A,H
4492 B5         01560           OR      L               ; CHECK FOR ZERO
4493 CA244B     01570           JP      Z,BADPRM        ; NOTHING THERE!
4496 CD164B     01580           CALL    GETLNO          ; GET INSERT LINE NO.
4499 1844       01590           JR      CHKLIN
                01600
449B            01610 FNDARG    EQU     $
                01620
                01630 ; SCAN THE COMMAND ENTRY FROM LEFT TO RIGHT LOOKING FOR
                01640 ; THE FIRST OCCURRENCE OF EITHER ":" OR "," TO DETERMINE
                01650 ; WHETHER SINGLE OR MULTIPLE LINES, RESPECTIVELY, ARE TO
                01660 ; BE MOVED.
                01670
449B 7E         01680           LD      A,(HL)          ; SCAN FOR DELIMITERS
449C FE3A       01690           CP      ':'
449E 280A       01700           JR      Z,PARSE         ; SINGLE LINE MOVE
44A0 FE2C       01710           CP      ','
44A2 2806       01720           JR      Z,PARSE         ; MULTIPLE LINE MOVE
44A4 23         01730           INC     HL
44A5 10F4       01740           DJNZ    FNDARG
44A7 C3244B     01750           JP      BADPRM          ; MUST HAVE "," OR ":"
                01760
44AA            01770 PARSE     EQU     $
44AA 47         01780           LD      B,A             ; DELIMITER TO "B" REG
44AB CD094B     01790           CALL    GETPRM          ; GET FIRST LINE NO.
44AE C5         01800           PUSH    BC              ; SAVE DELIMITER
44AF CDDA4A     01810           CALL    FNDLIN          ; FIND RECORD IN BUFFER
44B2 C2BB4C     01820           JP      NZ,NOLINE       ; NO SUCH LINE
44B5 F1         01830           POP     AF              ; DELIMITER
44B6 FE2C       01840           CP      ','
44B8 2805       01850           JR      Z,GETLST        ; MULTIPLE LINE NOS.
44BA 226244     01860           LD      (LSTLIN),HL     ; SAME LAST LINE NO.
44BD 1818       01870           JR      SAVLIN          ; SAVE PARAMS
                01880
44BF            01890 GETLST    EQU     $
                01900
                01910 ; EXTRACT THE LAST NO. OF THE RANGE OF LINES TO MOVE
                01920
44BF E5         01930           PUSH    HL
44C0 063A       01940           LD      B,':'           ; DELIMITER
44C2 CD094B     01950           CALL    GETPRM          ; GET LAST LINE NO.
44C5 D1         01960           POP     DE              ; FIRST LINE NO. TO DE
44C6 D5         01970           PUSH    DE
44C7 CDC24B     01980           CALL    CPHLDE          ; CHECK LINE NO. SEQUENCE
44CA DA244B     01990           JP      C,BADPRM        ; BAD PARAMETERS
44CD CDDA4A     02000           CALL    FNDLIN          ; FIND LAST RECORD
44D0 C2BB4C     02010           JP      NZ,NOLINE       ; NO SUCH LINE
44D3 226244     02020           LD      (LSTLIN),HL     ; LAST LINE NO.
44D6 E1         02030           POP     HL
                02040
44D7            02050 SAVLIN    EQU     $
44D7 226044     02060           LD      (CURLIN),HL     ; FIRST LINE NO.
                02070
```

```
                02080 ; GET THE INSERT LINE NO. FROM THE COMMAND ENTRY
                02090
44DA CDC74A     02100         CALL    LSTPRM          ; PARAMETERS AFTER ":"
44DD 2011       02110         JR      NZ,SKPTST       ; LINE DOESN'T EXIST
                02120
44DF            02130 CHKLIN  EQU     $
44DF EB         02140         EX      DE,HL
44E0 2A1741     02150         LD      HL,(INCR)       ; LINE NO. INCREMENT
44E3 19         02160         ADD     HL,DE           ; CALCULATE NEXT LINE NO.
44E4 DA3B4A     02170         JP      C,BADNO         ; LINE NO. TOO LARGE
44E7 11FAFF     02180         LD      DE,0FFFAH
44EA CDC24B     02190         CALL    CPHLDE
44ED D23B4A     02200         JP      NC,BADNO        ; LINE NO. TOO LARGE
                02210
44F0            02220 SKPTST  EQU     $
                02230
                02240 ; TEMPORARILY MODIFY THE ADDRESS OF EDTASM'S CALL TO THE
                02250 ; GETIPT ROUTINE REPLACING IT WITH THE ADDRESS OF A
                02260 ; ROUTINE TO EXTRACT THE DATA FROM THE TEXT BUFFER AND
                02270 ; MOVE IT INTO THE INPUT BUFFER AS THOUGH IT HAD BEEN
                02280 ; KEYED IN.
                02290
44F0 E5         02300         PUSH    HL              ; SAVE THE NEW LINE NO.
44F1 212645     02310         LD      HL,EXTRCT
44F4 22834A     02320         LD      (PROGSW),HL     ; MODIFY EDTASM
                02330
                02340 ; UTILIZE EDTASM'S INSERT ROUTINE TO CHECK FOR END OF
                02350 ; BUFFER, LINE NO. OVERLAP, ETC. AND INSERT THE LINE
                02360 ; INTO THE TEXT BUFFER AT THE PROPER LOCATION.
                02370
44F7 CD5C4A     02380         CALL    INSERT          ; INSERT THE RECORD
44FA 3A6644     02390         LD      A,(SWITCH)      ; CHECK THE DELETE OPTION
44FD B7         02400         OR      A
44FE 280F       02410         JR      Z,CHKEND        ; DON'T DELETE IT
                02420
                02430 ; USING THE LINE NO. FIND THE RECORD JUST MOVED (ADDRESS
                02440 ; HAS PROBABLY CHANGED) AND DELETE IT.
                02450
4500 2A6444     02460         LD      HL,(OLDLIN)     ; LINE ALREADY MOVED
4503 CDDA4A     02470         CALL    FNDLIN          ; FIND THE ADDRESS
4506 60         02480         LD      H,B
4507 69         02490         LD      L,C             ; LINE ADDRESS
4508 ED5B1D41   02500         LD      DE,(NXTPTR)     ; NEXT LINE ADDRESS
450C CDEA4C     02510         CALL    DELETE
                02520
450F            02530 CHKEND  EQU     $
                02540
                02550 ; SETUP FOR NEXT LINE TO BE INSERTED AND CHECK TO SEE IF
                02560 ; THE RANGE (IF ANY) IS FINISHED.
                02570
450F E1         02580         POP     HL              ; LAST INSERT LINE NO.
4510 CDDA4A     02590         CALL    FNDLIN          ; RE-ESTABLISH ADDRESS
4513 ED431141   02600         LD      (CURPTR),BC     ; STORE THE ADDRESS
4517 E5         02610         PUSH    HL
4518 ED5B6044   02620         LD      DE,(CURLIN)     ; NEXT LINE TO MOVE
451C 2A6244     02630         LD      HL,(LSTLIN)     ; FINAL LINE TO MOVE
451F CDC24B     02640         CALL    CPHLDE
4522 E1         02650         POP     HL
4523 30BA       02660         JR      NC,CHKLIN
4525 C9         02670         RET                     ; ALL DONE MOVING
                02680
4526            02690 EXTRCT  EQU     $
                02700
                02710 ; ROUTINE TO MOVE A LINE FROM THE TEXT BUFFER TO THE
                02720 ; EDTASM INPUT BUFFER AND MAKE IT LOOK AS THOUGH IT
                02730 ; HAD BEEN KEYED IN.
                02740
4526 2A6044     02750         LD      HL,(CURLIN)     ; GET LINE NO. TO MOVE
4529 226444     02760         LD      (OLDLIN),HL     ; SAVE IT FOR DELETE
452C CDDA4A     02770         CALL    FNDLIN          ; GET THE ADDRESS
```

Program continued

```
452F C2BB4C   02780          JP       NZ,NOLINE        ; NO SUCH LINE NO.
4532 60       02790          LD       H,B              ; ADDRESS TO HL FOR MOVE
4533 69       02800          LD       L,C
4534 112741   02810          LD       DE,BUFFER        ; POINT TO INPUT BUFFER
4537 D5       02820          PUSH     DE
4538 23       02830          INC      HL               ; SKIP OVER THE LINE NO.
4539 23       02840          INC      HL
453A 7E       02850          LD       A,(HL)           ; GET THE LINE LENGTH
453B 32AA41   02860          LD       (BUFLTH),A       ; SET INPUT BUFFER LENGTH
453E 23       02870          INC      HL               ; POINT TO DATA
453F 0600     02880          LD       B,0
4541 4F       02890          LD       C,A              ; SET UP LENGTH FOR MOVE
4542 CDC14A   02900          CALL     MOVE             ; RECORD TO INPUT BUFFER
4545 AF       02910          XOR      A
4546 12       02920          LD       (DE),A           ; SET TERMINATOR BYTE
              02930
              02940 ; GET THE LINE NO. FOR THE NEXT LINE THAT MAY BE MOVED
              02950
4547 7E       02960          LD       A,(HL)
4548 23       02970          INC      HL
4549 66       02980          LD       H,(HL)
454A 6F       02990          LD       L,A
454B 226044   03000          LD       (CURLIN),HL      ; NEXT LINE NO.
              03010
454E E1       03020          POP      HL               ; RESTORE BUFFER POINTER
454F 0680     03030          LD       B,80H            ; MAXIMUM LINE SIZE
4551 CD294E   03040          CALL     DISPLY           ; DISPLAY LINE ON CRT
4554 CD3747   03050          CALL     LINEFD
              03060
              03070 ; RESTORE THE MODIFIED EDTASM CALL STATEMENT
              03080
4557 214149   03090          LD       HL,GETIPT
455A 22834A   03100          LD       (PROGSW),HL
455D C9       03110          RET
              03120
              03130
              03140 ; PATCH TO MODIFY REFERENCE TO VIDEO DCB:
              03150
460B          03160          ORG      460BH
460B 1D40     03170          DEFW     401DH            ; ROM VIDEO DCB
              03180
              03190
              03200 ; PATCH TO MODIFY "B" COMMAND AND ITS ASSOCIATED ADDRESS:
              03210
492F          03220          ORG      492FH
492F 4D       03230          DEFB     'M'              ; REPLACES "B" COMMAND
4930 6744     03240          DEFW     START            ; "MOVE" ROUTINE ADDRESS
              03250
0000          03260          END
00000 TOTAL ERRORS
```

APPENDIX

Appendix A

Appendix B

APPENDIX A

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I, follow this procedure:

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P.*

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX B

Glossary

A

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200–450 nanoseconds for TRS-80 RAM.

accumulator—the main register(s) in a microprocessor used for arithmetic, shifting, logical, and other operations.

accuracy—generally, the quality or freedom from mistake or error; the extent to which the results of a calculation or a measurement approach the true value of the actual quantities.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

A/D converter—analog to digital converter. See D/A converter.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications, generally on large-scale computers.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—the process of adjusting components of a system for proper interrelationships, including adjustments and synchronization for the components in a system. For the TRS-80, this usually applies to cassette heads and disk drives.

alphanumerics—refer to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

alternating current—ac. Electric current that reverses direction periodically, usually many times per second.

ALU—Arithmetic Logic Unit.

analog—the representation of a physical variable by another variable insofar as the proportional relationships are the same over some specified range.

AND—a Boolean logic function. Two operators are tested and, if both are true, the answer is true. Truth is indicated by a high bit, or 1 in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately. A bit-by-bit logical operation which produces a one in the result bit only if both operand bits are ones.

anode—in a semiconductor diode, the terminal toward which electrons flow from an external circuit; the positive terminal.

APL—A Programming Language; a popular and powerful high-level mathematical language with extensive symbol manipulation.

argument—any of the independent variables accompanying a command.

Arithmetic Logic Unit—ALU. The section of a microprocessor which performs arithmetic functions such as addition or subtraction and logic functions such as ANDing.

array—a collection of data items arranged in a meaningful pattern such as rows and columns which allow the collection and retrieval of data.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—software that translates operational codes into their binary equivalents on a statement-for-statement basis.

assembly language—a symbolic computer language that is translated by an assembler program into machine language, the numeric codes that are equivalent to microprocessor instructions.

B

backup—1) refers to making copies of all software and data stored externally; 2) having duplicate hardware available.

base—the starting point for representation of a number in written form, where numbers are expressed as multiples of powers of the base value.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing in which many of the same types of jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud—1) a unit of data transmission speed equal to the number of code elements (bits) per second; 2) a unit of signaling speed equal to the number of discrete conditions or signal events per second.

baud rate—a measure of the speed at which serial data is transmitted electronically. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

BCD—binary coded decimal. The 4-bit binary notation in which individual decimal digits (0 through 9) are represented by 4-bit binary numerals; e.g., the number 23 is represented by 0010 0011 in the BCD notation.

bias—a dc voltage applied to a transistor control electrode to establish the desired operating point.

bidirectional bus—a bus structure used for the two-way transmission of signals, that is, both input and output.

bidirectional printer—a printer capable of printing both left-to-right and right-to-left. Data is prestored in a fixed-size buffer.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

binary digit—the two digits, 0 and 1, used in binary notation. Often shortened to bit.

bi-stable—two-state

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

bit position—the position of a binary digit within a byte or larger group of binary digits. Bit positions in the Model I, II, III, and Color Computer are numbered from right to left, zero through N. This number corresponds to the power of two represented.

Boolean algebra—a mathematical system of logic first identified by George Boole, a 19th century English mathematician. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. On some machines it is keyed in, and on others it is in read only memory (ROM). Using this program is called booting or cold-starting the system.

bps—bits per second.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software or hardware.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specially designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

card reader—a device for reading information from punched cards.

cassette recorder—a magnetic tape recording and playback device for entering or storing programs.

cathode—in a semiconductor diode, the terminal from which electrons flow to an external circuit; the negative terminal.

character—a single symbol that is represented inside the computer by a specific code.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for the block of data.

chip—the shaped and processed semiconductor die mounted on a substrate to form a transistor or other semiconductor device.

circuit—a conductor or system of conductors through which an electric current may flow.

circuit card—a printed circuit board containing electronic components.

clear—to return a memory to a non-programmed state, usually represented as 0 or OFF (empty).

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed or frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business-Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

compiler—software that will convert a program written in a high-level language to binary code, on a many-for-one basis.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

computer interface—a device designed for data communication between a central computer and another unit such as a programmable controller processor.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

conductor—a substance, body, or other medium that is suitable to carry an electric current.

constant—a value that doesn't change.

CPU—central processing unit. The circuitry that actually performs the functions of the instruction set.

CRT—cathode ray tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to a read/write section of tape.

cursor—a visual movable pointer used on a CRT by the programmer to indicate where an instruction is to be added to the program. The cursor is also used during editing functions.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

daisy wheel—a printer type which has a splined character wheel.

data—general term for numbers, letters, symbols, and analog quantities that serve as information for computer processing.

data base—refers to a series of programs each having a different function, but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

debug—to remove bugs from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

diagnostic program—a test program to help isolate hardware malfunctions in the programmable controller and application equipment.

digital—the representation of data in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

digital circuit—an electronic network designed to respond at input voltages at one level, and similarly, to produce output voltages at one level.

diode—a device with an anode and a cathode which permits current flow in one direction and inhibits current flow in the other direction.

direct current—dc. Electric current which flows in only one direction; the term designates a practically non-pulsating current.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system—DOS. The system software that manipulates the data to be sent to the disk controller.

dividend—the number that is divided by the divisor. In A/B, A is the dividend.

divisor—the number that "goes into" the dividend in a divide operation. In A/B, B is the divisor.

DMA—direct memory access. A process where the CPU is disabled or

bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

dot-matrix printer—instead of each letter having a separate type head (like a typewriter), a single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to manufacture.

downtime—the time when a system is not available for production due to required maintenance.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrical Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editors.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Erasable Programmable Read Only Memory. A read only memory in which stored data can be erased by ultraviolet light or other means and reprogrammed bit-by-bit with appropriate voltage pulses.

Exclusive OR—a bit-by-bit logical operation which produces a one bit in the

result only if one or the other (but not both) operand bits is a one.

execution—the performance of a specific operation such as would be accomplished through processing one instruction, a series of instructions, or a complete program.

execution cycle—a cycle during which a single instruction of one specific operation is performed.

execution time—the total time required for the execution to actually occur.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

exponent—the power to which a floating-point number is raised.

F

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

field-effect transistor—FET. A transistor in which the resistance of the current path from the source to drain is modulated by applying a transverse electric field between grid or gate electrodes; the electric field varies the thickness of depletion layers between the gates, thereby reducing the conductance.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occurred.

flip-flop—a bi-stable device that assumes either of two possible states such that the transition between the states must be accomplished by electronic switching.

floating-point number—a standard way of representing any size number in computers. Floating-point numbers contain a fractional portion (mantissa) and power of two (exponent) in a form similar to scientific notation.

flowcharting—a method of graphically displaying program steps, used to develop and define an algorithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

full duplex—a mode of data transmission that is the equivalent of two paths—one in each direction simultaneously.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function. A circuit having an output and a multiplicity of inputs, so designed that the output is energized only when a certain combination of pulses is present at the inputs.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

ground—a conducting path, intentional or accidental, between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

H—a suffix for hexadecimal, e.g., 4FFFH.

half duplex—data can flow in both directions, but not simultaneously. See duplex.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Handshaking requires clocking pulses on both ends of the communications line. Contrast with buffer.

hard copy—a printout; any form of printed document such as a ladder diagram, program listing, paper tape, or punched cards.

hardware—refers to any physical piece of equipment in a computer system.

hex—hexadecimal.

hexadecimal—representation of numbers in base sixteen by use of the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming language which is CPU-independent and closely resembles English.

high order—see most significant bit.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

human engineering—usually refers to designing hardware and software with ease of use in mind.

I

IC—integrated circuit.

immediate—addressing mode in which the address of the information that an operation is supposed to act upon immediately follows the operation code.

increment—to increase, usually by one. See decrement.

indexed—addressing mode where the information is addressed by a specified value, or by the value in a specified register.

indirect—addressing mode in which the address given points to another address, and the second address is where the information actually is.

input devices—devices such as limit switches, pressure switches, push buttons, etc., that supply data to a programmable controller. These discrete inputs are two types: those with common return, and those with individual returns (referred to as isolated inputs). Other inputs include analog devices and digital encoders.

instruction—a command or order that will cause a computer to perform one particular operation.

integer variable—a BASIC variable type. It can hold values of −32,768 through +32,767 in two-byte two's complement notation.

integrated circuit—IC. An interconnected array of active and passive elements integrated with a single semiconductor substrate or deposited on the substrate and capable of performing at least one electronic circuit function. See chip.

intelligent terminal—a terminal with a CPU and a certain amount of memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compiler.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

iteration—one pass through a given set of instructions.

J

jack—a socket, usually mounted on a device, which will receive a plug (generally mounted on a wire).

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

L

language—a set of symbols and rules for representing and communicating information (data) among people, or between people and machines.

large scale integration—LSI. Any integrated circuit which has more than 100 equivalent gates manufactured simultaneously on a single slice of semiconductor material.

least significant bit—the rightmost bit in a binary value, representing 2^0.

least significant byte—refers to the lowest position digit of a number. The rightmost byte of a number or character string.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory. The last data pushed onto the stack is the first popped out.

light emitting diode—LED. A semiconductor diode that displays alphanumeric characters when supplied with a specified voltage.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

line—in communications, describes cables, telephone lines, etc., over which data is transmitted to and received from the terminal.

line printer—a high-speed printing device that prints an entire line at one time.

location—a storage position in memory.

logic—a means of solving complex problems through the repeated use of simple functions which define basic concepts. Three basic logic functions are AND, OR, and NOT.

logic diagram—a drawing which represents the logic functions AND, OR, NOT, etc.

logic level—the voltage magnitude associated with signal pulses representing ones and zeroes (1s and 0s) in binary computation.

logical shift—a type of shift in which an operand is shifted right or left, with a zero filling the vacated bit position.

loop—a set of instructions that executes itself continuously. If the programmer has the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value, the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

lsb—see least significant bit.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

mantissa—the fractional portion of a floating-point number.

matrix—a two-dimensional array of circuit elements, such as wires, diodes, etc., which can transform a digital code from one type to another.

memory—the hardware that stores data for use by the CPU. Each piece of

data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

metal oxide semiconductor—MOS. A metal insulator semiconductor structure in which the insulating layer is an oxide of the substrate material; for a silicon substrate the insulator is silicon oxide.

micro electronics—refers to circuits built from miniaturized components and includes integrated circuits.

microprocessor—an electronic computer processor section implemented in relatively few IC chips (typically LSI) which contain arithmetic, logic, register, control, and memory functions.

microsecond—us. One millionth of a second: 1×10^{-6} or 0.000001 second.

millisecond—ms. One thousandth of a second: 10^{-3} or 0.001 second.

minuend—the number from which the subtrahend is subtracted.

mixed number—a number consisting of an integer and fraction as, for example, 4.35 or (binary) 1010.1011.

mnemonic—a short, alphanumeric abbreviation used to represent a machine-language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

module—an interchangeable plug-in item containing electronic components which may be combined with other interchangeable items to form a complete unit.

monitor—1) a CRT; 2) a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

MOS—see metal oxide semiconductor.

MOSFET—metal oxide semiconductor field effect transistor.

most significant bit—the leftmost bit in a binary value, representing the highest-order power of two. In two's complement notation, this bit is the sign bit.

most significant byte—the highest-order byte. In the multiple-precision number A13EF122H, A1H is the most significant byte.

msb—see most significant byte.

multiplexing—a method allowing several sets of data to be sent at different times over the same communication lines, yet all of the data can be used simultaneously after the final set is received. For example, several LED displays, each requiring four data lines, can all be written to with only one group of four data lines. The same concept is used with communication lines.

multiplicand—the number to be multiplied by the multiplier.

multiplier—the number that is multiplied against the multiplicand. The number "on the bottom."

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

nanosecond—one billionth of a second.

nesting—putting one loop inside another. Some computers limit the number of loops that can be nested.

noise—extraneous signals; any disturbance which causes interference with the desired signal or operation.

non-volatile memory—a memory that does not lose its information while its power supply is turned off.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1). This is the one's complement.

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0–7.

OEM—Original Equipment Manufacturer.

off-line—describes equipment or devices which are not connected to the communications line.

off-the-shelf—a term referring to software. A generalized program that can be used by many computer owners. It is mass produced and can be bought off-the-shelf.

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

operands—the numeric values used in the add, subtract, or other operation.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

output—the current, voltage, power, driving force, or information which a circuit or a device delivers. The terminals or other places where a circuit or device can deliver energy.

output devices—devices such as solenoids, motor starters, etc., that receive data from the programmable controller.

overflow—a condition that exists when the result of an add, subtract, or other arithmetic operation is too large to be held in the number of bits allotted.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

page—refers to a 256 (2 to the 8th power) word block of memory. How large a word depends on the computer. Most micros are eight-bit word machines. Many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously. Contrast with serial.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even, the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

parity bit—an additional bit added to a memory word to make the sum of the number of 1s in a word always even or odd as required.

parity check—a check that tests whether the number of 1s in an array of binary digits is odd or even.

PC board—see printed circuit board.

peripheral devices—a generic term for equipment attached to a computer, such as keyboards, disk drives, cassette tapes, printers, plotters, speech synthesizers.

permutation—arrangements of things in definite order. Two binary digits have four permutations: 00, 01, 10, and 11.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters. Generally used for CAI.

PL/1—an acronym for Programming Language 1. A programming language used by very large computers. It incorporates most of the better features from other programming languages. Its power comes from the fact that bits can be manipulated from the high-level language.

plotter—a device that can draw graphs and curves and is controlled by the computer through an interface.

port—a single addressable channel used for communications.

positional notation—representation of a number where each digit position represents an increasingly higher power of the base.

precision—the number of significant digits that a variable or number format may contain.

printed circuit board—a piece of plastic board with lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

processor—a unit in the programmable controller which scans all the inputs and outputs in a predetermined order. The processor monitors the status of the inputs and outputs in response to the user-programmed instructions in memory, and it energizes or de-energizes outputs as a result of the logical comparisons made through these instructions.

product—the result of a multiply.

program—a sequence of instructions to be executed by the processor to control a machine or process.

PROM—Programmable Read Only Memory. A memory device that is written to once and from then on acts like a ROM.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

punched-card equipment—peripheral devices that enable punching or reading paper punched cards that hold character or binary data.

Q

quotient—the result of a divide operation.

R

RAM—acronym for Random Access Memory. An addressable LSI device used to store information in microscopic flip-flops or capacitors. Each may be set to an ON or OFF state, representing logical 1 or 0. This type of

memory is volatile, that is to say, memory is lost while power is off, unless battery backup is used.

read—to sense the presence of information in some type of storage, which includes RAM memory, magnetic tape, punched tape, etc.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a fast-access memory location in the microprocessor. Used for holding intermediate results and for computation in machine language.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

remainder—the amount of dividend remaining after a divide has been completed.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the machine will try, but the data is not remembered.

rounding—the process of truncating bits to the right of a bit position and adding zero or one to the next higher bit position based on the value to the right. Rounding the binary fraction 1011.1011 to two fractional bits, for example, results in 1011.11.

RPG—an acronym for Report Program Generator. A language for business that primarily reads data from cards and prints reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

S

scaling—multiplying a number by a fixed amount so that a fraction can be processed as an integer value.

scientific notation—a standard form for representing any size number by a mantissa and power of ten.

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricity. Transistors and integrated circuits are made from semiconductive material and are called semiconductors.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. Contrast with parallel.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (−) and 0 is positive (+).

signed numbers—numbers that may be either positive or negative.

significant bits—the number of bits in a binary value after leading zeros have been removed.

significant digit—a digit that contributes to the precision of a number. The number of significant digits is counted beginning with the digit contributing the most value, called the most significant digit, and ending with one contributing the least value, called the least significant digit.

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

software—refers to the programs that can be run on a computer.

solid state devices (semiconductors)—electronic components that control electron flow through solid materials such as crystals; e.g., transistors, diodes, integrated circuits.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

SPOOL—acronym for Simultaneous Peripheral Output, On-Line. Used to overlap processing, typically, with printing.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

storage—see memory.

subroutine—a routine within a program that ends with an instruction to return program flow to where it was before the routine began. This routine is used many times from many different places in the program, and the subroutine allows you to write the code for that routine only once. Similar to a macro.

subtrahend—the number that is subtracted from the minuend.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system—a collection of units combined to work as a larger integrated unit having the capabilities of all the separate units.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

tape reader—a unit which is capable of sensing data from punched tape.

TeletypeTM—a peripheral electromechanical device for entering or outputting a program or data in either a punched paper tape or printed format.

text editor—see word processor.

time sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

transistor—an active component of an electronic circuit consisting of a small

block of semiconducting material to which at least three electrical contacts are made, usually two closely spaced rectifying contacts and one ohmic (non-rectifying) contact; it may be used as an amplifier, detector, or switch.

transistor-transistor logic—TTL. A logic circuit containing two transistors, for driving large output capacitances at high speed. A family of integrated circuit logic. (Usually 5 volts is high or 1, and 0 volts is low or 0; 5V = 1, 0V = 0).

truncation—the process of dropping bits to the right of a bit position. Truncating the binary fraction 1011.1011 to a number with fraction of two bits, for example, results in 1011.10.

truth table—a table defining the results for several different variables and containing all possible states of the variables.

TTL—see transistor-transistor logic.

TTY—an abbreviation for Teletype.

two's complement—a standard way of representing positive and negative numbers in microcomputers.

U

unsigned numbers—numbers that may be only positive; absolute numbers.

utility—a program designed to aid the programmer in developing other software.

UV erasable PROM—an ultraviolet erasable PROM is a programmable read-only memory which can be cleared (set to 0) by exposure to intense ultraviolet light. After being cleared, it may be reprogrammed.

V

variable—a labeled entity that can take on any value.

volatile memory—a memory that loses its information if the power is removed from it.

von Neumann, John (1903–1957)—mathemetician. He put the concept of games, winning strategy, and different types of games into mathematical formulae. He also advanced the concept of storing the program in memory as opposed to having it on tape.

W

weighted value—the numerical value assigned to any single bit as a function of its position in the code word.

word—a grouping or a number of bits in a sequence that is treated as a unit and is stored in one memory location. If the CPU works with 8 bits, then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

zero flag—a bit in the microprocessor used to record the zero/non-zero status of the result of a machine-language instruction.

zero page—refers to the first page of memory.

INDEX

INDEX

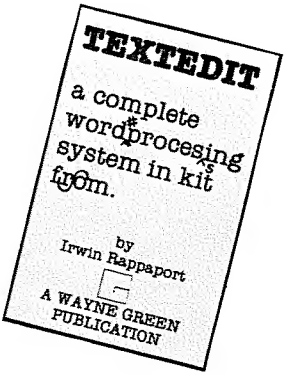
INDEX COMPILED BY NAN MCCARTHY

Wayne Green Books

TEXTEDIT

A Complete Word Processing System in Kit Form

by Irwin Rappaport



Word processing systems can cost hundreds of dollars and, even when you've bought one, it probably won't do everything you want.

TEXTEDIT is an inexpensive word processor that you can adapt to suit your needs, from writing form letters to large texts. It is written in modules, so you can load and use only those portions that you need. Included are modules that perform:

- right justification
- ASCII upper/lowercase conversion
- one-key phrase entering
- complete editorial functions
- and much more!

TEXTEDIT is written in TRS-80* Disk BASIC, and the modules are documented in the author's clear writing style. Not only does Irwin Rappaport explain how to use **TEXTEDIT**; he also explains programming techniques implemented in the system.

TEXTEDIT is an inexpensive word processor that helps you learn about BASIC programming. It is written for TRS-80 Models I and III with TRSDOS 2.2/2.3 and 32K.

ISBN 0-88006-050-6

90 pages

$9.97

WAYNE GREEN BOOKS
Division of Wayne Green Inc.
Peterborough, NH 03458

FOR TOLL FREE ORDERING:
1-800-258-5473

*TRS-80 and TRSDOS are trademarks of Radio Shack division of Tandy Corp.

Wayne Green Books

THE NEW WEATHER SATELLITE HANDBOOK

by Dr. Ralph E. Taggart WB8DQT

The New Weather Satellite Handbook is a completely updated and revised edition of the best-selling ***Weather Satellite Handbook***, containing all the information on the most sophisticated and effective spacecraft now in orbit. Dr. Taggart has written this book to serve both the experienced amateur satellite enthusiast and the newcomer. ***The New Weather Satellite Handbook*** is an introduction to satellite watching, providing all the information required to construct a complete and highly effective ground station. Not just ideas, but solid hardware designs and all the instructions necessary to operate the equipment are included. For the thousands of experimenters who are operating stations, the book details all procedures necessary to modify their equipment for the new series of spacecraft. An entire chapter is devoted to microcomputers and their use in the weather satellite station, focussing particularly on the Radio Shack TRS-80* microcomputers.

ISBN 0-88006-015-8 136 pages **$8.95**

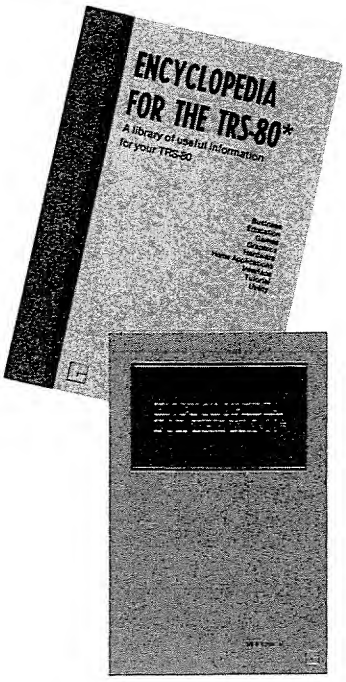
WAYNE GREEN BOOKS
Division of Wayne Green Inc.
Peterborough, NH 03458
FOR TOLL FREE ORDERING:
1-800-258-5473

*TRS-80 is a trademark of Radio Shack division of Tandy Corp.

Wayne Green Books

ENCYCLOPEDIA FOR THE TRS-80*

A 10-VOLUME SERIES



What's the key to getting the most from your TRS-80*? No, it isn't disk drives or printers or joysticks. It's information. Without a continual supply of information and ideas, you cannot realize the full potential of your TRS-80.

The ***Encyclopedia for the TRS-80**** is a 10-volume reference work of programs and articles that have been carefully selected to help you make the most of your microcomputer. You can think of the ***Encyclopedia*** as an extension of the documentation that came with your TRS-80. Each book contains material on programming techniques, business, hardware, games, tutorials, education, utilities, interfacing, graphics and home applications.

HARDCOVER EDITION **$19.95** per volume
SOFTCOVER EDITION **$10.95** per volume

ENCYCLOPEDIA LOADER™

The editors at Wayne Green Books created the **Encyclopedia Loader™** to help you maximize the use of your microcomputing time. By a special arrangement with Instant Software™, you can purchase selected programs from each volume of the ***Encyclopedia for the TRS-80**** in cassette form. Your ***Encyclopedia*** provides the essential documentation, but now you'll be able to load the programs instantly. With the **Encyclopedia Loader™**, you'll save hours of keyboard time and eliminate the aggravating search for typos.

$14.95 per cassette

WAYNE GREEN BOOKS
Division of Wayne Green Inc.
Peterborough, NH 03458

FOR TOLL FREE ORDERING:
1-800-258-5473

*TRS-80 is a trademark of Radio Shack division of Tandy Corp.

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

$10.95 ISBN 0-88006-045-X